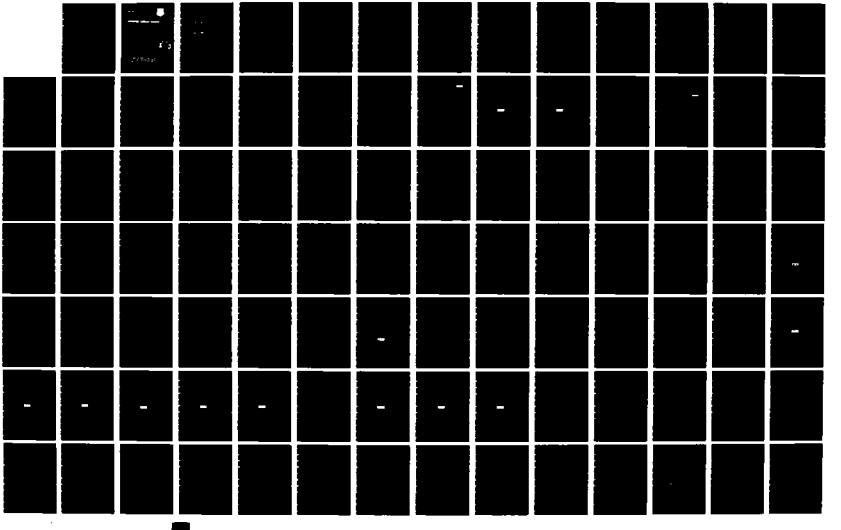
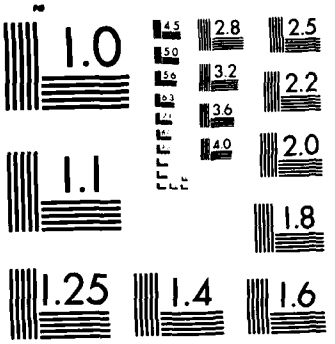
AD-A162 457 SPECIFICATION TECHNOLOGY GUIDEBOOK(U) BOEING AEROSPACE 1/3
CO SEATTLE WA D R ADDLEMAN ET AL AUG 85
RADC-TR-85-135 F30602-84-C-0073

UNCLASSIFIED F/G 9/2 NL





RADC-TR-85-125
Final Technical Report
August 1985

(12)

SPECIFICATION TECHNOLOGY GUIDEBOOK

Boeing Aerospace Company

David R. Addleman, Margaret J. Davis and P. Edward Presson

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

[redacted] has been reviewed and is approved for publication.

APPROVED: William E. Rzepka

WILLIAM E. RZEPKA
Project Engineer

APPROVED: Raymond P. Urtz, Jr.

RAYMOND P. URTZ, JR.
Technical Director
Command & Control Division

FOR THE COMMANDER: Richard W. Poulliot

RICHARD W. POULIOT
Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COEE) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document requires that it be returned.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

AD-A162 457

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS N/A
2a SECURITY CLASSIFICATIC ↓ AUTHORITY N/A	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b DECLASSIFICATION / DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) N/A	5. MONITORING ORGANIZATION REPORT NUMBER(S) RADC-TR-85-135
6a NAME OF PERFORMING ORGANIZATION Boeing Aerospace Company 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Rome Air Development Center (COEE)
6c ADDRESS (City, State, and ZIP Code) P.O. Box 3999 Seattle WA 98124	7b ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700
8a NAME OF FUNDING / SPONSORING ORGANIZATION Rome Air Development Center 8b. OFFICE SYMBOL (If applicable) COEE	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F30602-84-C-0073
8c. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. 62702F; PROJECT NO. 5581; TASK NO. 22; WORK UNIT ACCESSION NO. 13

11 TITLE (Include Security Classification)
SPECIFICATION TECHNOLOGY GUIDEBOOK

12 PERSONAL AUTHOR(S)
David R. Addleman, Margaret J. Davis, P. Edward Presson

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM Mar 84 TO Mar 85	August 1985	242

16. SUPPLEMENTARY NOTATION
N/A

17 COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Software Specification Methodology Guidelines Software Specification Tools Software Requirements Methodology, (See Reverse)
09	02		

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

The specification Technology Guidebook is designed for Air Force technical managers. Using its guidelines, software development project managers can select methodologies and tools at the front-end of the software development life cycle that will not only benefit software projects during system requirements, software requirements, and design phases, but also during the remaining life-cycle phases.

The field of specification technology is in continual expansion. New methodology and tools enter the marketplace weekly, while older ones mature or are adapted to accommodate different computer hardware, CPU's, or languages. For this reason the guidelines are constructed in modular form for easy inclusion of new methodologies and tools or revised descriptions of old ones.

Further, the guidelines are designed for use by Air Force technical managers on projects contracted to companies of varying software engineering practices. In general, the approach

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL William E. Rzepka	22b TELEPHONE (Include Area Code) (315) 330-4063 22c OFFICE SYMBOL RADC (COEE)

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

incorporates MIL-STD-490 and meets the requirements of all DOD and service software standards. The life cycle information is in accordance with AFR 800-14 and DOD-STD-2167 standards.

The method by which the user selects methodologies and tools is based on the use of rating tables. These tables have been carefully constructed to permit a compact representation of many selection considerations. A more complete discussion of these considerations can be found in the Final Report of RADC Contract F30602-84-C-0073, and can be studied as a companion volume to this guidebook.

The guidelines presented in this volume are the culmination of surveys of Air Force missions, current technical literature (e.g., journals, conference proceedings, and textbooks), discussions with specification technology developers, hands-on testing of many methodology and tools software packages, hours of analysis, and some trial and error approaches. These guidelines provide the Air Force with a simplified approach to specification technology selection that will, for the majority of new projects, allow the technical manager to select the methodology and tools that best suit his needs.

18. Subject Terms (Continued).

Software Design Methodology

PREFACE

The Specification Technology Guidebook provides guidelines for the selection of requirements and design specification methodologies appropriate to various software development environments and various types of software. The guidelines cover the requirements analysis, architectural and detailed design phases. These guidelines are incorporated in a table-driven format that define increasingly thorough and formal levels of specification based on a software project's *significance level.* Significance level measures the relative importance of an individual project based on considerations of quality, software, and project.

The guidebook provides summary descriptions of specification methodologies. It includes a method for selecting automated tools to support the selected methodologies. It includes typical paragraphs that can be included in Air Force software development statements-of-work to specify the use of specification methodologies by the contractor during the requirements analysis and design phases of a contracted development.

Three example problems for C^3I software development projects are included. A primary consideration imposed on each example is compatibility with the Ada* programming language. The other considerations used for system requirements and design of the C^3I problems were derived from actual requirements set forth in C^3I RFP's, and working knowledge of the requirements for C^3I software and system projects gained by Boeing Aerospace engineers during the last decade.

* Ada is a trademark of the U.S. Department of Defense (Ada Joint Program Office).

TABLE OF CONTENTS

TABLE OF CONTENTS - continued

TABLE OF CONTENTS - continued

LIST OF FIGURES

LIST OF FIGURES - continued

ABBREVIATIONS

AD	Armament Division
AFR	Air Force Requlation
AFTI	advanced fighter technology integration
ASD	Aeronautical Systems Division
ASSM	Abstract System Semantic Model
ATD	air crew training device
ATO	Air Tasking Order
BMO	Ballistic Missile Office
BSD	Berkeley Software Distribution
CAFMS	Computer Assited Air Force Management System
CINCSAC	Commander-in-Chief Strategic Air Command
COM	computer output microfiche
CPCD	computer program development plan
CPCI	computer program configuration item
CPU	central processing unit
CRT	interactive terminal
CSCI	computer software configuration item
CSOC	Consolidated Space Operations Center
DCDS	Distributed Computing Design System
DDC	Dansk Datamatik Center
DDL	distributed design language
DoD	Department of Defense
DSSD	Data Structured System Design
ELINT	electronic intelligence
GMB	Graph Model of Behavior
HDM	Hierarchical Design Methodology
HOL	higher order language
HSL	hierarchical specification language
ICBM	intercontinental ballistic missile
IUS	inertial upper stage
IV&V	independent verification and validation
JINTACCS	Joint Interoperable Tactical Air Command and Control System
JSD	Jackson System Design
JSTPS	Joint Strategic Target Planning Staff
LRU	line replacement unit
MDL	modular design language
MIL	military
OFP	operation flight programs
OSL	overall significance level

ABBREVIATIONS - continued

PAISLey	Process-Oriented, Applicative, Interpretable Specification Language
pdl	program design language
PROM	programmable read-only memory
RADC	Rome Air Development Center
RFP	request for proposal
RSL	requirements statement language
SADT	Structured Analysis and Design Technique
SARA	System Architect's Apprentice
SCF	Satellite Control Facility
SCR	Software Cost Reduction project
SD	Space Division
SDL	software development laboratory
SILTF	System Integration Laboratory and Test Facility
SIOP	Single Integrated Operation Plan
SL	significance level
SPO	System Program Office
SREM	Software Requirements Engineering Methodology
SRU	shop replaceable unit
STD	standard
TAC	Tactical Air Command
TACC	Tactical Air Control Center
TACS	Tactical Air Control System
TDI	transition diagram interpreter
TLS	top level specification
TRD	Test Requirements Document
TSL	test specification language
UUT	unit under test
VDM	Vienna Development Method

1.0 SPECIFICATION TECHNOLOGY GUIDEBOOK

1.1 INTRODUCTION

The Specification Technology Guidebook is designed for Air Force technical managers. Using the guidelines herein, software development project managers can select methodologies and tools, at the front-end of the software development life cycle, that will not only benefit software projects during system requirements, software requirements, and design phases, but also during the remaining life-cycle phases.

The field of specification technology is in continual expansion; new methodology and tools enter the marketplace weekly, while older ones mature or are adapted to accommodate different computer hardware, CPU's, or languages. For this reason, the guidelines are constructed in modular form, for easy inclusion of new methodologies and tools, or revised descriptions of old ones.

Further, the guidelines are designed for use by Air Force technical managers on projects contracted to companies of varying software engineering practices. In general, our approach incorporates MIL-STD 490 and meets the requirements of all DoD and Service software standards. The life cycle information is in accordance with AFR 800-14 and DoD-STD-SDS standards.

The method by which the user selects methodologies and tools is based on the use of rating tables found in section 2.0. These tables have been carefully constructed to permit a compact representation of many selection considerations. A more complete discussion of these considerations can be found in the Final Report of RADC Contract F30602-84-C-0073, and can be studied as a companion volume to this guidebook.

The guidelines presented in this volume are the culmination of surveys of Air Force missions, current technical literature (e.g., journals, conference proceedings, and textbooks), discussions with specification technology developers, hands-on testing of many methodology and tools software packages, hours of analysis, and some trial and error approaches. These guidelines provide the Air Force with a simplified approach to specification technology selection that will, for the majority of new projects, allow the technical manager to select the methodology and tools that best suit his needs.

1.2 OUTLINE OF SPECIFICATION TECHNOLOGY GUIDEBOOK

The Specification Technology Guidebook comprises six major sections and six appendices, as shown in figure 1-1. A brief summary of contents is presented in the following paragraphs.

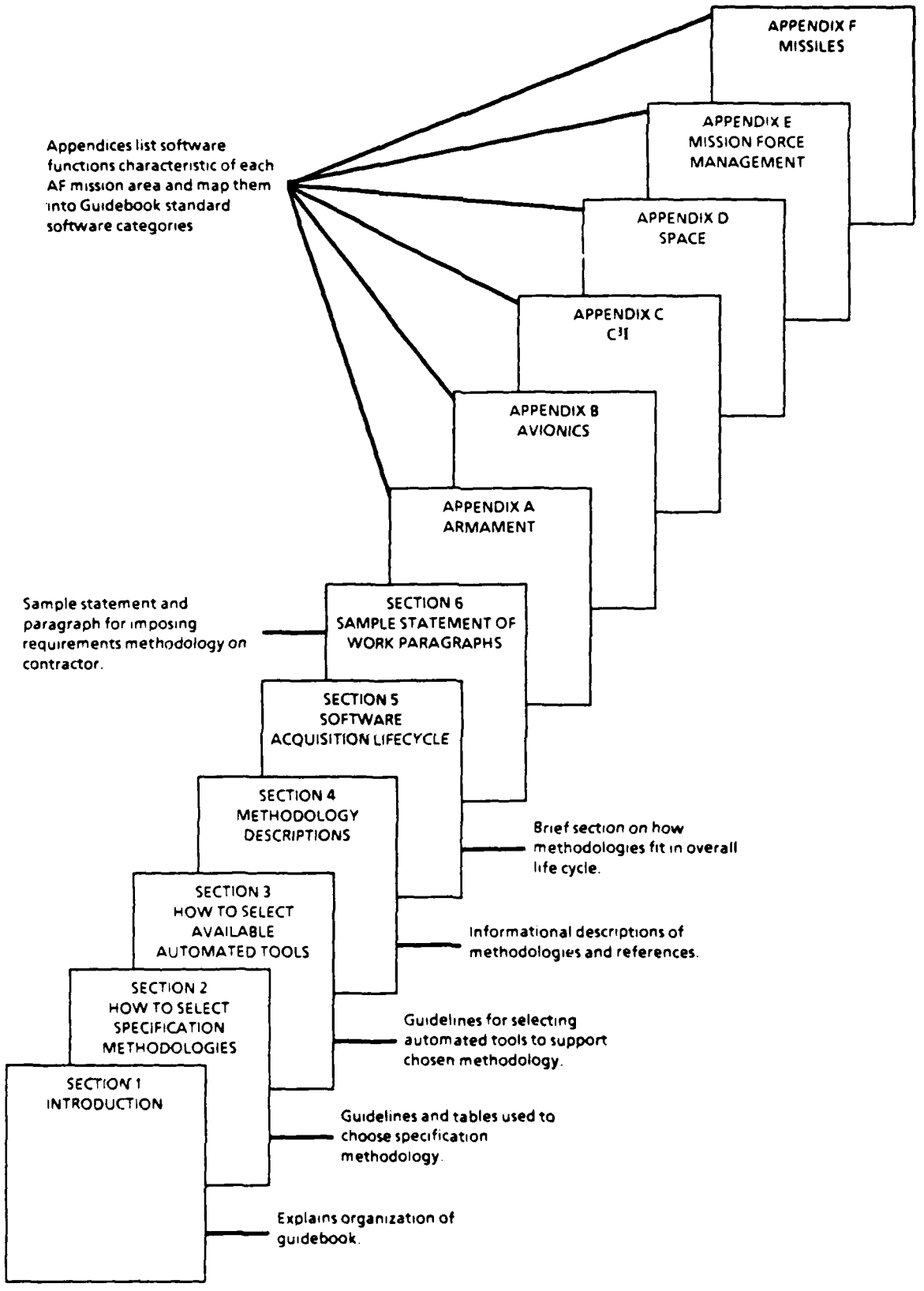


Figure 1-1. Guidebook Organization

Section 1.0 introduces the guidebook, stating objectives, describing outline and content, and discussing applications.

Section 2.0 describes how an Air Force technical manager may select a suitable methodology for his project by means of the known project requirements and the guidelines and tables provided. Three possible selection paths are given, along with examples which step the reader through each selection path.

Section 3.0 describes how the Air Force technical manager may select automated tools that will be compatible with his methodology.

Section 4.0 provides detailed descriptions of methodologies and tools.

Section 5.0 describes the software acquisition life-cycle standards most likely to be encountered by the Air Force technical manager, briefly discussing how these relate to specification methodologies listed in the guidebook.

Section 6.0 provides sample SOW paragraphs for guiding the AF technical manager in writing statements of work for specification technologies.

The appendices describe six Air Force mission areas: armament; avionics; command, control, communication, and intelligence (C^3I); missiles; space; and mission/force management. Each appendix lists software functions characteristic of the computer programs developed within that mission. Each function is classified according to software categories used in section 2.0.

1.3 APPLICATIONS OF THE GUIDEBOOK

The principal purpose of this guidebook is to provide the Air Force technical manager with a means to select a methodology and compatible tools for his future software project. Additionally, the guidebook can be used in preparing a Statement of Work, and for evaluating proposals.

All three applications of the guidebook make use of the tables and selection paths of section 2.0 in selecting appropriate specification methodologies.

The guidebook was prepared for a user with a general technical background, who may be unfamiliar with specific system/software requirements and design technologies.

1.4 CONSIDERATIONS USED IN RATING REQUIREMENTS AND DESIGN METHODOLOGIES AND TECHNIQUES

Several considerations were used during development of a matrix that rates the specification technologies techniques evaluated and discussed in the guidelines. These are described in the following paragraphs.

1.4.1 Concept Expressibility

Whether a specification methodology is suitable for a particular development project depends primarily upon the concepts specifiable by the requirements methodology and the approach to structuring the architectural design taken by the design methodology. A requirements specification will be inadequate if the methodology cannot model the important features of the system to be developed. A design specification will be inadequate if the structuring technique couples software modules too strongly to facilitate change, simply because the technique ignored a more important set of relationships.

1.4.2 Degree of Automated Support

The suitability of a specification methodology for a particular development project depends upon the significance of the project and the amount of computer support a methodology provides. *It would not be cost-effective to use a very powerful* methodology/toolset like SREM/REVS for a project whose end product was a prototype of a software development environment tool (e.g., a calendar program). It is the more significant, complex, and time-consuming projects that will benefit most from more powerful methodologies.

2.0 HOW TO SELECT SPECIFICATION METHODOLOGIES

This section presents guidelines for use by the project manager in selecting a specification technology for a future project.

2.1 INTRODUCTION TO METHODOLOGY SELECTION

The first part of this section provides the guidelines for selecting a methodology and supporting tools for a project, by system category and with full consideration given to life cycle phase. By following the selection process in this section, the project manager will be guided through the maze of current specification technologies and tools to arrive at a final selection that fulfills his needs. The process of final methodology selection will also aid the technical manager in assessing his project requirements and help him evaluate his needs during the entire project life cycle.

The second part of this section illustrates how the guidelines and tables are used in a C^3I example. The objective is to define a front-end environment for developing C^3I systems that is compatible with the Ada programming language.

2.2 METHODOLOGY SELECTION PATHS

The guidelines provide a choice of three paths, depending on which software acquisition life cycle the project is in when the manager wants to perform methodology selection. A Path is provided for use at each of the following:

1. Path 1. At concept definition (i.e., in the Requirements Phase of the life cycle).

2. Path 2. After requirements analysis is complete (i.e., in the Design Phase of the life cycle).

3. Path 3. When the project dictates the use of certain capabilities (i.e., independent of the life cycle).

The guidelines outline steps along each Path, providing tables for reference and a worksheet to fill out along the way. Paths 1 and 2 are similar; the first is used to select a methodology using requirements data and the second is used to select a methodology using design data. Path 3 is shorter, and is used when a project manager knows that the final methodology must contain specific capabilities.

The selection process for Paths 1 and 2 comprise four steps, as follows:

1. Step 1. Choose the overall significance level (OSL) of the project.

2. Step 2. Select the best-fit software category.

3. Step 3. Designate candidate methodologies.

4. Step 4. Compare scores for candidate methodologies.

In accomplishing the four steps, above, the project manager is required to reference tables and enter data on a worksheet. Once the candidates are designated (Step 3), final methodology selection is straightforward.

Section 2.3 contains step-by-step instructions for reading the tables, filling out the worksheet, and selecting a software specification methodology following Path 1. Section 2.4 contains the instructions for following Path 2. Section 2.5 contains instructions for Path 3.

2.3 REQUIREMENTS METHODOLOGY SELECTION - Path 1

An outline of the Path is shown in figure 2-1. The steps are discussed individually in the following paragraphs.

2.3.1 Step 1 -- Choose the Overall Significance Level (OSL)

The first step in determining the correct methodology for a project is to choose an Overall Significance Level (OSL) value for that project. Two items are needed: the Significance Level Table, figure 2-2 and the Methodology Selection Worksheet, figure 2-3. A completed example worksheet is shown in figure 2-4.

The Significance Level (SL) Table is divided into five major columns: *Project Considerations, Software Considerations, Quality Considerations, Software Examples,* and *Significance Level.* Under each major column are sub-columns, each labeled by a consideration. For example, under the major column *Project Considerations* are three sub-columns *Cost, Criticality,* and *Schedule.* Similarly, the major columns *Software Considerations* and *Quality Considerations* have three and four sub-columns, respectively. The *Software Examples* column provides example software products for use as a guide in assessing significance levels. The *Significance Level* column contains the SL numeric values that are assigned to the capabilities, based on the considerations.

The Methodology Selection Worksheet (also called "the worksheet") is divided into four columns: *Considerations, SL, Weight,* and *Product.* Note that the rows under the *Considerations* column correspond to the consideration sub-columns in the Significance Level Table (figure 2-2).

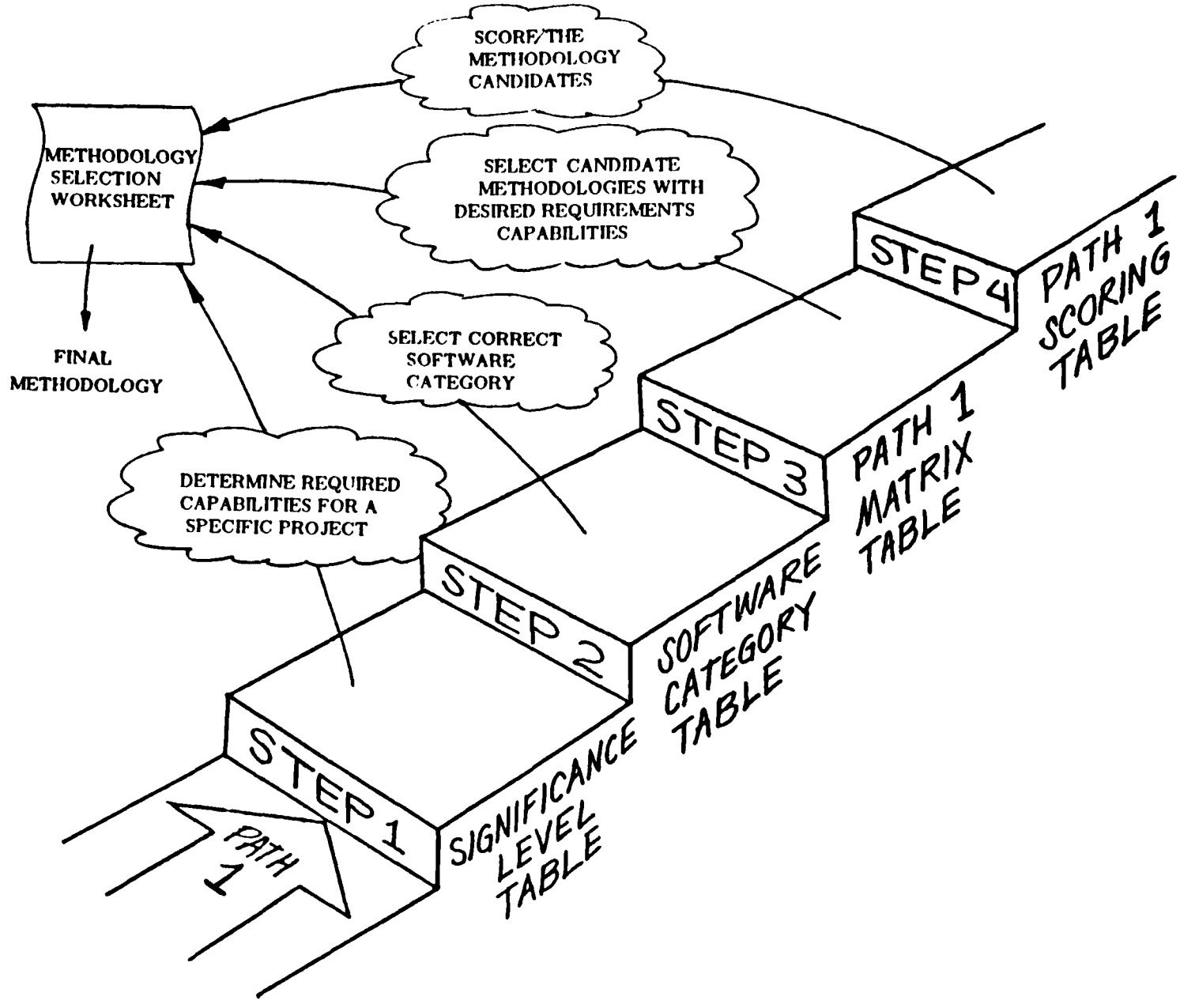


Figure 2-1 Path 1 Overview

SIGNIFICANCE LEVEL TABLE

Project Considerations			Software Considerations			Quality Considerations				Software	Significance
Cost	Criticality	Schedule	Complexity	Devt Formality	Software Utility	Reliability	Correctness	Maintainability	Verifiability		
Low Budget, emphasis on min cost	No criticality assignment	Tight Schedule	Straightforward solution; easy to checkout	Few defined requirements; informal development; used locally	One-shot; Prototype; Test S/W; Demo S/W	Respond correctly to nominal conditions	Functionality met; constraints ignored	None expected	Documentation is source code	Test Gener; conversion table, trade study simulated	0
Normal cost constraints	Nuisance Impact, no Mission impact	Some schedule constraints	Moderate Complexity	Normal to Strong Contractor Controls, informal reviews	Ground Based S/W, Data Reduction; Mission Prep S/W	Faults corrected periodically; temporary workarounds provided	Functionality and constraints met	Predict impact of changes	Source code documentation updated	Editor, compiler, mission simulation, environmental simulator	1
Some Cost Flexibility	Mission Impact	Normal Schedule Constraints	Greater Complexity	Strong Contractural Controls, Formal Reviews	Real-time Avionics, C3 s/w, C3I	Faults removed ASAP	Implementation validated against design specification	Impact of changes somewhat localised	Full complement of documentation; design documentation updated too	AWACS, ALCM, PMALS, C3I, AVIONICS MISSION PLANNING	2
Cost not predominant factor, relatively unconstrained	Nuclear, flight crew safety	Additional requirements will not impact schedule	Difficult Problem, Complex Solution, Hard to Validate	Rigid Contractual Controls Over Development	Highly Critical Applications, Possible Catastrophic Results	No faults	Design validated against requirements specification	Extent of changes optimally localised	Requirements through source code documentation always up-to-date	Nuclear Controls; critical software	3

A fault is an undesirable response to anomalous conditions.

None of the present mature software development methodologies enforce documentation of enhancements or changes at any level.

Figure 2-2 Significance Level Table

METHODOLOGY SELECTION WORKSHEET

SOFTWARE TO BE ACQUIRED _____

LIFECYCLE PHASE .....____REQUIREMENTS ____DESIGN

CONSIDERATIONS	SL (0, 1, 2, 3) FIGURE 2-2	WEIGHT (1 = NORMAL)	PRODUCT (WEIGHT X SL)
COST			
CRITICALITY			
SCHEDULE			
COMPLEXITY			
DEVELOPMENT FORMALITY			
SOFTWARE UTILITY			
RELIABILITY			
CORRECTNESS			
MAINTAINABILITY			
VERIFIABILITY			
SUM			
OSL = SUM OF PRODUCT / SUM OF WEIGHT =			
SOFTWARE CATEGORY			
CANDIDATE METHODOLOGIES			
KEY			
SCORE			

Figure 2-3 Methodology Selection Worksheet

METHODOLOGY SELECTION WORKSHEET

SOFTWARE TO BE ACQUIRED EXAMPLE
LIFECYCLE PHASE ..... X REQUIREMENTS ___ DESIGN

CONSIDERATIONS	SL (0, 1, 2, 3) FIGURE 2-2	WEIGHT (1 = NORMAL)	PRODUCT (WEIGHT X SL)
COST	1	2	2
CRITICALITY	1	1	1
SCHEDULE	3	0	0
COMPLEXITY	1	1	1
DEVELOPMENT FORMALITY	1	1	1
SOFTWARE UTILITY	0	1	0
RELIABILITY	2	1	2
CORRECTNESS	2	1	2
MAINTAINABILITY	2	1	2
VERIFIABILITY	1	1	1
SUM	[redacted]	10	12
OSL = SUM OF PRODUCT / SUM OF WEIGHT = $\frac{12}{10}$ = 1.20 Round down → 1			
SOFTWARE CATEGORY			6
CANDIDATE METHODOLOGIES			
KEY	D	F	
SCORE	12	21	

Figure 2-4 Example Methodology Selection Worksheet

Additional copies of the Methodology Selection Worksheet are located at the end of this section. If all copies have already been torn out, then figure 2-3, Methodology Selection Worksheet, can be reproduced, and returned to the guidebook.

Entries for the worksheet are defined as follows:

1. CONSIDERATIONS -- the column of considerations whose names correspond to sub-columns in the Significance Level Table (figure 2-2).

2. SL -- the column of significance level values (0 through 3) relating to the considerations in the Significance Level Table.

3. WEIGHT -- the column of weighted values assigned to the considerations.

4. PRODUCT -- the column of products of SL's and weights.

5. SUM -- two sums are used: the sum of the weight column and the sum of the product column.

6. OSL -- the Overall Significance Level for the project, obtained by dividing the product sum by the weight sum.

7. SOFTWARE CATEGORY -- the numerical category obtained from the Software Categories Table (figure 2-6) in Step 2.

8. KEY -- the alphabetic key that represents a methodology, obtained in Step 3.

9. SCORE -- the numeric score for a methodology, obtained in Step 4.

In Step 1 the user begins filling out the worksheet, using the SL table and his knowledge of the software project. To begin, the user asks: *What is the significance of cost for the project?* Under the *Cost* column, four choices correspond to Significance Level numbers in the right-most column of the table. Thus, if *Normal cost constraints* were chosen as the most appropriate cost project consideration, the SL would be *1*. Figure 2-5, Example Use of Significance Table, illustrates this procedure. In this case, a *1* is placed under SL in the *Cost* row of the Methodology Selection Worksheet, as shown in figure 2-4. For each column in the table, the description is located that best fits the consideration for that column, then the corresponding SL number is written on the worksheet in the correct row under SL. At the end of this process, the SL column of the worksheet will contain ten values.

Next, the ten considerations are weighted by entering values in the *Weight* column. The weights reflect how relevant or important a single consideration is to a project. If all considerations are weighted equally, a *1* is entered in each row of the *Weight* column. If a consideration is critical, a higher weighting (e.g., *3)* is assigned. A reasonable range

Find Cost

SIGNIFICANCE LEVEL TABLE

Project Considerations			Software Considerations			Quality Considerations				Software	Significance
Cost	Criticality	Schedule	Complexity	Devt Formality	Software Utility	Reliability	Correctness	Maintainability	Verifiability		
Low Budget, emphasis on min cost	No criticality assignment	Tight Schedule	Straightforward solution, easy to checkout	Few defined requirements, informal development, used locally	One-shot; Prototype; Test S/W, Demo S/W	Respond correctly to nominal conditions	Functionality met; constraints ignored	None expected	Documentation in source code	Test Gener; conversion table, trade study simulated	0
Normal cost constraints	No significance Impact, no Mission impact	Some schedule constraints	Moderate Complexity	Normal to Strong Contractor Controls, informal reviews	Ground Based S/W; Data Reduction; Mission Prep S/W	Faults corrected periodically; temporary workarounds provided	Functionality and constraints met	Predict impact of changes	Source code documentation updated	Editor, compiler, model simulation, environmental simulator	1
Some Cost Flexibility	Mission Impact	Normal Schedule Constraints	Greater Complexity	Strong Contractural Controls, Formal Reviews	Real-time Avionics, C3 s/w, C3I	Faults removed ASAP	Implementation validated against design specification	Impact of changes somewhat localized	Full complement of documentation; design documentation updated too	AWACS, ALCM, PMALS, C3I, AVIONICS MISSION PLANNING	2
Cost not predominant factor, relatively unconstrained	Nuclear flight crew safety	Additional requirements will not impact schedule	Difficult Problem, Complex Solution, Hard to Validate	Rigid Contractual Controls Over Development	Highly Critical Applications, Possible Catastrophic Results	No faults	Design validated against requirements specification	Extent of changes optimally localized	Requirements through source code documentation always up-to-date	Nuclear Controls; critical software	3

Enter "1" on work sheet

Figure 2-5 Example Use of Significance Table

for weight values is 0 through 3. Since a weight of *0* implies that a consideration is not relevant to the project, a negative weight is meaningless. A *1* is *normal*; a *2* is *important*; and a *3* is *very important*. Numerical values above *3* lose significance, since the calculation is relatively insensitive to a single consideration's weight value. Weight assignment is subjective and dependent on the project manager's knowledge of the proposed project, but is valuable in emphasizing considerations. Initially, the project manager may feel more comfortable assigning a *1* as the weighting value, but as he gains experience, he will acquire a *feel* for how weighting influences final methodology selection.

The Product column is completed on a row-by-row basis by multiplying the SL by the Weight. For example, if the SL is *1* and the weight is *2* the product is 2 X 1 = 2. The *2* is entered in the Product column. Each row on the worksheet is processed similarly, resulting in entries for all considerations in all three columns.

The sum of the weight column is calculated and entered in the *Sum* row; the sum of the product column is calculated and entered in the *Sum* row. The Overall Significance Level (OSL) for the project is calculated as follows:

OSL = Product Sum / Weighting Sum

Thus, if the Product Sum = 12, and the Weighting Sum = 10, the OSL = 1.20. Since whole numbers are required in future steps, the fraction must be rounded up or down. In determining this, the consideration is located with the highest weighting and its corresponding SL noted. If the SL is 2 or greater, the OSL is rounded up; if 1 or less, rounded down. In our example, cost has the highest weight; the SL for cost is 1, so we round down. The final OSL becomes *1* and is entered on the worksheet.

2.3.2 Step 2 -- Select the Best-Fit Software Category

In Step 2 the user determines which software category best fits his proposed software project. In accomplishing this he uses the Software Categories Table, figure 2-6, and enters the result on the Methodology Selection Worksheet.

The Software Categories Table is a multi-page table of three columns: *Category, Characteristics*, and *Description*. The user reviews the 18 software categories, using his knowledge of the proposed software project, and selects the most relevant category.

The categories, characteristics, and descriptions in the tables are grouped by complexity and criticality in accordance with a survey of Air Force software engineering considerations, undertaken during the Software Test Handbook contract, F30602-82-C-0059.[1] A user finding difficulty in identifying an appropriate software category, should refer to

1. See Software Test Handbook Final Report (RADC-TR-84-53, Vol I) for details of the survey.

SOFTWARE CATEGORIES TABLE

Category	Characteristics	Description
(1) Arithmetic Based	Data oriented	Programs that do primarily arithmetic (e.g., payroll and wind tunnel data analysis) operations. A real-time environment is not necessary. Small, throwaway programs for preliminary analysis also fit in this category.
(2) Event control	Control-oriented processing	Does real-time processing of data resulting from external events. An example might be a computer program that processes telemetry data.
(3) Process control	Control-oriented processing	Receives data from an external source and issues commands to that source to control its actions based on the received data.
(4) Procedure control	Complex processing	Controls other software; for example, an operating system that controls execution of time-shared and batch computer programs.
(5) Navigation	Complex processing	Does computations and modeling to compute information required to guide an airplane from point of origin to destination.
(6) Flight Dynamics	Control-dominated complex processing	Uses the functions computed by navigation software and augmented by control theory to control the entire flight of an aircraft.

Figure 2-6 Software Categories Table (part 1 of 3)

continued		
Category	Characteristics	Description
(7) Orbital Dynamics	Control-dominated complex processing	Resembles navigation and flight dynamics software, but has the additional complexity required by orbital navigation, such as a more complex reference system and the inclusion of gravitational effects of other heavenly bodies.
(8) Message processing	Data-dominated complex processing	Handles input and output messages, processing the text or information contained therein.
(9) Diagnostic S/W	Data-oriented processing	Used to detect and isolate hardware errors in the computer in which it resides or in other hardware that can communicate with that computer.
(10) Sensor and signal processing	Control-dominated complex processing	Similar to that of message processing, except that it requires greater processing, analyzing, and transforming the input into a usable data processing format.
(11) Simulation	Complex, depending on entity being simulated	Used to simulate an environment, mission situation, other hardware, and inputs from these to enable a more realistic evaluation of a computer program or a piece of hardware.
(12) Database management	Data-oriented processing	Manages the storage and access of (typically large) groups of data. Such software can also often prepare reports in user-defined formats, based on the contents of the database.

Figure 2-6 Software Categories Table (part 2 of 3)

continued		
Category	Characteristics	Description
(13) Data Acquisition	Control-dominated complex processing	Receives information in real-time and stores it in some form suitable for later processing; for example, software that receives data from a space probe and files it for later analysis.
(14) Data presentation	Data-oriented	Formats and transforms data, as necessary, for convenient and understandable displays for humans. Typically, such displays would be for some screen presentation.
(15) Decision and planning aids	Data-dominated complex processing	Uses artificial intelligence techniques to provide an expert system to evaluate data and provide additional information and consideration for decision and policymakers.
(16) Pattern and image processing	Data-dominated complex processing	Used for computer image generation and processing. Such software may analyse terrain data and generate images based on stored data.
(17) Computer system Software	Data-oriented	Provides services to operational computer programs (i.e., problem-oriented).
(18) Software development tools	Data-oriented	Provides services to aid in the development of software (e.g., compilers, assemblers, static and dynamic analyzers).

Figure 2-6 Software Categories Table (part 3 of 3)

the appendix describing the pertinent Air Force mission and find the software function most similar to the proposed software project. The category number assigned to that software function can be used as the software category on the worksheet. The software category number is entered in the *Software Category* row of the worksheet (as shown in figure 2-4).

The categories are identical to those found in the Software Test Handbook (RADC-TR-84-53, Vol II), except the category formerly designated *Batch* has been renamed *Arithmetic-Based* to better convey its nature.

2.3.3 Step 3 -- Designate Candidate Methodologies

In Step 3 the user matches the capabilities of the methodologies against the capabilities he desires in a requirements methodology; i.e., the user knows which capabilities he'd like in a methodology and this step allows him to select those methodologies coming closest to having those capabilities. To accomplish this, the user will need the Path 1 Match Table, figure 2-7, and the Methodology Selection Worksheet.

A methodology that is a good candidate will have approximately the same pattern of entries (where *x* entries indicate the capabilities present in a methodology) in a row of the *Methodology* section of the Path 1 Match Table that the *Software Category* section has in the software category row. Note that a perfect match occurs when a row in the *methodology* section has at least the same number of *x*'s in the same columns as does the row in the *Software Category* section; i.e., a perfect match still exists if more than the needed matching *x*'s are contained in a *methodology* row. The key letters for the best candidates are entered on the worksheet.

For instance (see figure 2-8, Example Use of Path 1 Match Table), software category *6* needs a methodology with capabilities for state modeling, data flow modeling, control flow modeling, object modeling, and timing specification (i.e., these columns contain *x's).*

In the Methodology section, two methodologies have identical entry patterns, D and F. Thus, methodologies D and F become candidates for final selection and are so noted on the worksheet. Perfect matches may not exist between a software category's capabilities and the methodology capabilities. For example, the K methodology is close to matching and could be selected as a valid candidate, unless its absence of the state modeling capability would critically affect the project. The user must look for reasonable capability approximations in selecting candidate methodologies. He may select a set of candidates having either less or more capabilities than those desired.

In general, completing Step 3 means the user has selected several candidate methodolo-

Path 1 Match Table							
		Modeling Techniques				Performance Specification	
		State	Flow		Object	Timing	Accuracy
			Data	Control			
SOFTWARE CATEGORY	1		x		x		x
	2			x		x	
	3			x		x	
	4		x	x		x	
	5		x	x	x	x	x
	6	x	x	x	x	x	
	7	x	x	x	x	x	x
	8		x	x	x	x	
	9		x		x		x
	10	x	x	x	x	x	x
	11	x	x	x	x	x	
	12		x		x		x
	13	x	x	x	x	x	x
	14		x		x		x
	15	x	x	x	x		x
	16		x	x	x		x
	17		x	x	x	x	
	18		x		x		
METHODOLOGY	A		x	x	x		
	B	x			x		x
	C		x	x		x	x
	D	x	x	x	x	x	
	E	x			x	x	x
	F	x	x	x	x	x	x
	G	x			x		x
	H	x			x		x
	I	x	x		x		
	J	x			x	x	x
	K		x	x	x	x	
	L	x	x	x			

Figure 2-7 Path 1 Match Table

Path 1 Match Table

		Modeling Techniques				Performance Specification	
		State	Flow		Object	Timing	Accuracy
			Data	Control			
SOFTWARE CATEGORY	1		x		x		x
	2			x		x	
	3			x		x	
	4		x	x		x	
	5		x	x	x	x	x
	6	x	x	x	x	x	
	7	x	x	x	x	x	x
	8		x	x	x	x	
	9		x		x		x
	10	x	x	x	x	x	x
	11	x	x	x	x	x	
	12		x		x		x
	13	x	x	x	x	x	x
	14		x		x		x
	15	x	x	x	x		x
	16		x	x	x		x
	17		x	x	x	x	
	18		x		x		
METHODOLOGY	A		x	x	x		
	B	x			x		x
	C		x	x		x	x
	D	x	x	x	x	x	
	E	x			x	x	x
	F	x	x	x	x	x	x
	G	x			x		x
	H	x			x		x
	I	x	x		x		
	J	x			x	x	x
	K		x	x	x	x	
	L	x	x	x			

software category "6"

matches "D" & "F"

nearly matches "K"

Figure 2-8 Example Use of Path 1 Match Table

gies having capabilities critical to the project's life cycle phase and software category.

2.3.4 Step 4 -- Compare Scores for Candidate Methodologies

In Step 4 the user scores each candidate methodology and determines the final methodology for his proposed software project. To accomplish this, he will need the Methodology Selection Worksheet and one of four tables.

He begins by finding the proper methodology score table for his Path and OSL. On Path 1, he uses figure 2-9 if the OSL is *0*, figure 2-10 if the OSL is *1*, figure 2-11 if the OSL is *2*, and figure 2-12 if the OSL is *3*.

After accessing the correct table for his OSL, he locates the methodology letter under the *Methodology* column, then follows that row to the column under the *Software Category* number. See figure 2-13, Example Path 1 Methodology Scores for OSL=1. The intersection of the methodology row and the software category column contains the pre-calculated score for the candidate methodology. He enters this score in the *Score* row, under the candidate methodology key letter, on the worksheet.

The candidate methodology that best fits the proposed software project requirements will be the one with a final score nearest zero. That is, if methodology D has a final score of 12 and methodology F has a final score of 21, then methodology D is the best fit. A methodology score found in the four OSL tables indicates how the methodology compares to a fictional *ideal* methodology. A positive score means the methodology provides more support to the project than nominally desired; a negative score means the methodology provides less support to the project than nominally desired; a score of zero means a methodology provides the support nominally desired.

2.4 DESIGN METHODOLOGY SELECTION - Path 2

Path 2 is similar to Path 1, except that Step 3 matches desirable design phase capabilities (instead of the requirements phase capabilities of Path 1) against the Path 2 match table, and the scoring of the methodologies in Step 4 makes use of a separate set of design phase OSL tables.

An outline of the design methodology selection Path is shown in figure 2-14. The Steps are discussed individually in the following paragraphs.

2.4.1 Step 1 -- Choose the Overall Significance Level (OSL)

Path 1
Methodology Scores (OSL = 0)

Methodology	Software Category																	
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
A	24	23	23	24	25	30	27	28	24	27	30	24	27	24	22	29	28	25
B	38	34	34	37	39	41	46	40	38	46	41	38	46	38	31	41	40	37
C	26	27	27	26	27	32	30	39	26	30	32	26	30	26	22	29	39	25
D	32	31	31	32	33	39	37	35	32	37	39	32	37	32	30	37	35	31
E	31	28	28	28	30	36	36	35	31	36	36	31	36	31	24	34	35	30
F	44	42	42	44	47	48	51	44	44	51	48	44	51	44	37	47	44	40
G	27	19	19	20	25	28	29	27	27	29	28	27	29	27	19	31	27	25
H	47	40	40	41	46	49	50	48	47	50	49	47	50	47	33	52	48	45
I	28	21	21	22	22	31	29	30	28	29	31	28	29	28	22	29	30	26
J	33	36	36	36	39	40	43	39	33	43	40	33	43	33	26	36	39	30
K	40	45	45	45	47	53	51	51	40	51	53	40	51	40	34	47	51	42
L	40	37	37	42	39	46	47	42	40	47	46	40	47	40	32	43	42	38

Figure 2-9 Path 1 Methodology Scores for OSL=0

Path 1
Methodology Scores (OSL = 1)

Methodology	Software Category																	
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
A	2	1	1	1	1	3	-1	3	2	-1	3	2	-1	2	2	4	3	4
B	16	12	12	14	15	14	18	15	16	18	14	16	18	16	11	16	15	16
C	4	5	5	3	3	5	2	4	4	2	5	4	2	4	2	4	4	4
D	10	9	9	9	9	12	9	10	10	9	12	10	9	10	10	12	10	10
E	9	6	6	5	6	9	8	10	9	8	9	9	8	9	4	9	10	9
F	22	20	20	21	23	21	23	19	22	23	21	22	23	22	17	22	19	19
G	5	-3	-3	-3	1	1	1	2	5	1	1	5	1	5	-1	6	2	4
H	25	18	18	18	22	22	22	23	25	22	22	25	22	25	13	27	23	24
I	6	-1	-1	-1	-2	4	1	5	6	1	4	6	1	6	2	4	5	5
J	11	14	14	13	15	13	15	14	11	15	13	11	15	11	6	11	14	9
K	18	23	23	22	23	26	23	26	18	23	26	18	23	18	14	22	26	21
L	18	15	15	19	15	19	19	17	18	19	19	18	19	18	12	18	17	17

Figure 2-10 Path 1 Methodology Scores for OSL=1

Path 1
Methodology Scores (OSL = 2)

Methodology	Software Category																	
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
A	-20	-21	-21	-22	-23	-24	-29	-22	-20	-29	-24	-20	-29	-20	-18	-21	-22	-17
B	-6	-10	-10	-9	-9	-13	-10	-10	-6	-10	-13	-6	-10	-6	-9	-9	-10	-5
C	-18	-17	-17	-20	-21	-22	-26	-21	-18	-26	-22	-18	-26	-18	-18	-21	-21	-17
D	-12	-13	-13	-14	-15	-15	-19	-15	-12	-19	-15	-12	-19	-12	-10	-13	-15	-11
E	-13	-16	-16	-18	-18	-18	-20	-15	-13	-20	-18	-13	-20	-13	-16	-16	-15	-12
F	0	-2	-2	-2	-1	-6	-5	-6	0	-5	-6	0	-5	0	-3	-3	-6	-2
G	-17	-25	-25	-26	-23	-26	-27	-23	-17	-27	-26	-17	-27	-17	-21	-19	-23	-17
H	3	-4	-4	-5	-2	-5	-6	-2	3	-6	-5	3	-6	3	-7	2	-2	3
I	-16	-23	-23	-24	-26	-23	-27	-20	-16	-27	-23	-16	-27	-16	-18	-21	-20	-16
J	-11	-8	-8	-10	-9	-14	-13	-11	-11	-13	-14	-11	-13	-11	-14	-14	-11	-12
K	-4	1	1	-1	-1	-1	-5	1	-4	-5	-1	-4	-5	-4	-6	-3	1	0
L	-4	-7	-7	-4	-9	-8	-9	-8	-4	-9	-8	-4	-9	-4	-8	-7	-8	-4

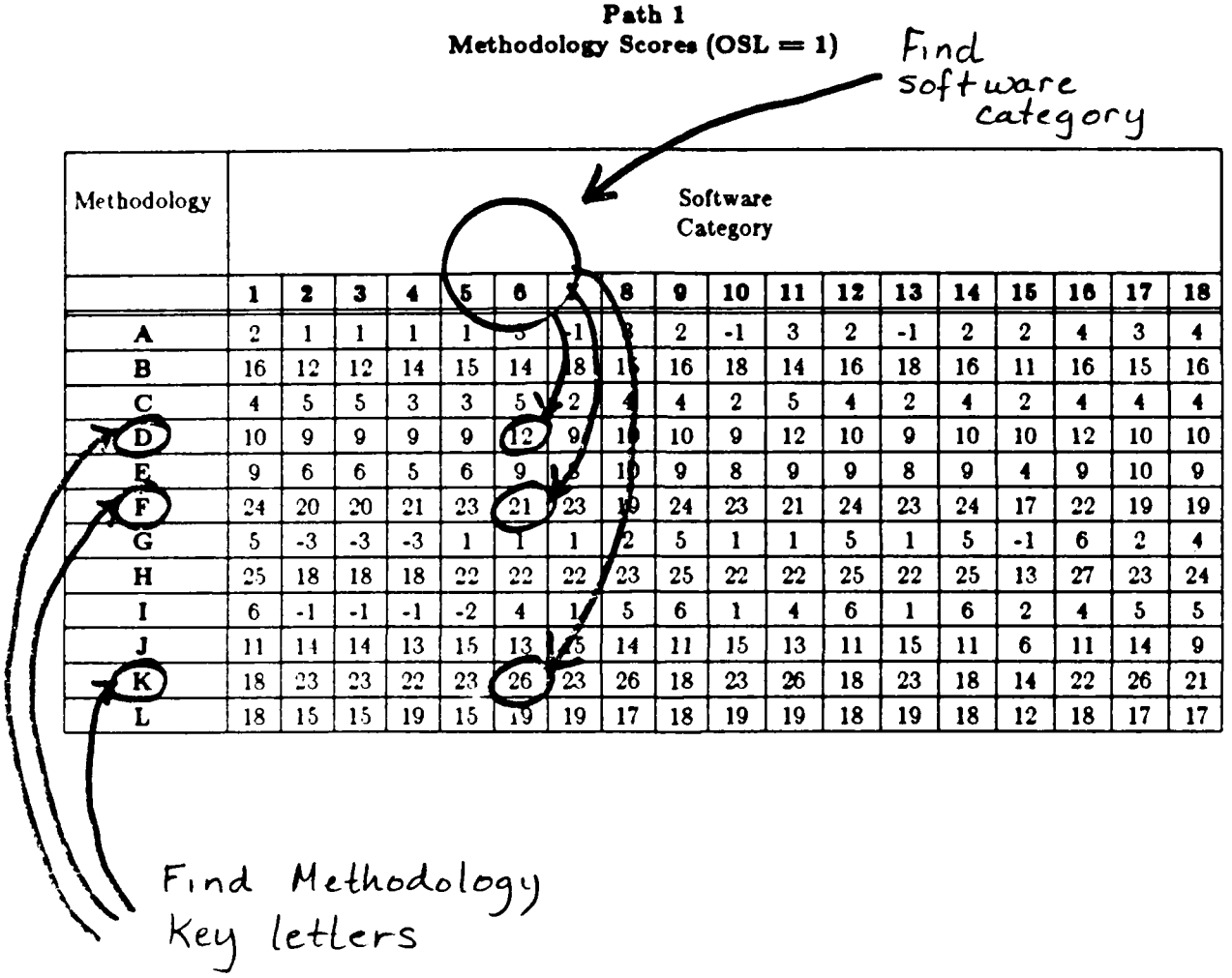
Figure 2-11 Path 1 Methodology Scores for OSL=2

Path 1
Methodology Scores (OSL = 3)

Methodology	Software Category																	
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
A	-42	-43	-43	-45	-47	-51	-56	-47	-42	-56	-51	-42	-56	-42	-38	-46	-47	-38
B	-28	-32	-32	-32	-33	-40	-38	-35	-28	-38	-40	-28	-38	-28	-29	-34	-35	-26
C	-40	-39	-39	-43	-45	-49	-54	-46	-40	-54	-49	-40	-54	-40	-38	-46	-46	-38
D	-34	-35	-35	-37	-39	-42	-47	-40	-34	-47	-42	-34	-47	-34	-30	-38	-40	-32
E	-35	-38	-38	-41	-42	-45	-48	-40	-35	-48	-45	-35	-48	-35	-36	-41	-40	-33
F	-22	-24	-24	-25	-25	-33	-33	-31	-22	-33	-33	-22	-33	-22	-23	-28	-31	-23
G	-39	-47	-47	-49	-47	-53	-55	-48	-39	-55	-53	-39	-55	-39	-41	-44	-48	-38
H	-19	-26	-26	-28	-26	-32	-34	-27	-19	-34	-32	-19	-34	-19	-27	-23	-27	-18
I	-38	-45	-45	-47	-50	-50	-55	-45	-38	-55	-50	-38	-55	-38	-38	-46	-45	-37
J	-33	-30	-30	-33	-33	-41	-41	-36	-33	-41	-41	-33	-41	-33	-34	-39	-36	-33
K	-26	-21	-21	-24	-25	-28	-33	-24	-26	-33	-28	-26	-33	-26	-26	-28	-24	-21
L	-26	-29	-29	-27	-33	-35	-37	-33	-26	-37	-35	-26	-37	-26	-28	-32	-33	-25

Figure 2-12 Path 1 Methodology Scores for OSL=3

Path 1
Methodology Scores (OSL = 1)



Methodology	Software Category																	
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
A	2	1	1	1	1	3	-1	3	2	-1	3	2	-1	2	2	4	3	4
B	16	12	12	14	15	14	18	15	16	18	14	16	18	16	11	16	15	16
C	4	5	5	3	3	5	2	4	4	2	5	4	2	4	2	4	4	4
D	10	9	9	9	9	12	9	10	10	9	12	10	9	10	10	12	10	10
E	9	6	6	5	6	9	6	10	9	8	9	9	8	9	4	9	10	9
F	24	20	20	21	23	21	23	19	24	23	21	24	23	24	17	22	19	19
G	5	-3	-3	-3	1	1	1	2	5	1	1	5	1	5	-1	6	2	4
H	25	18	18	18	22	22	22	23	25	22	22	25	22	25	13	27	23	24
I	6	-1	-1	-1	-2	4	1	5	6	1	4	6	1	6	2	4	5	5
J	11	14	14	13	15	13	15	14	11	15	13	11	15	11	6	11	14	9
K	18	23	23	22	23	26	23	26	18	23	26	18	23	18	14	22	26	21
L	18	15	15	19	15	19	19	17	18	19	19	18	19	18	12	18	17	17

Figure 2-13 Example Use of Path 1 Methodology Tables

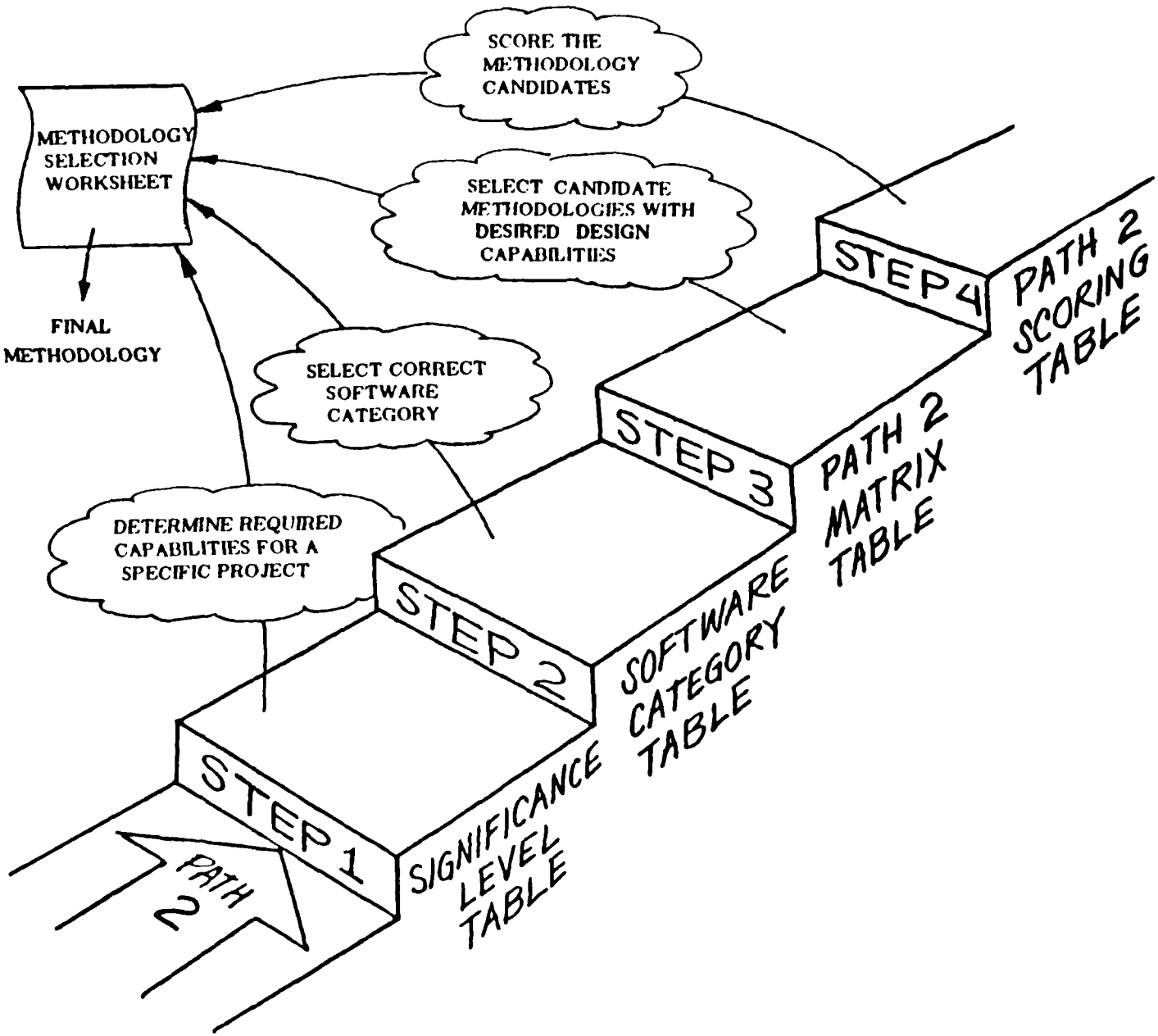


Figure 2-14 Path 2 Overview

The first step in using design-phase considerations is to determine an Overall Significance Level (OSL) value for that project. The approach to completing Step 1 is identical to the one taken in Path 1 for the requirements phase. The same table and worksheet are used: the Significance Level Table, figure 2-2 and the Methodology Selection Worksheet, figure 2-3.

In Step 1 the user begins filling out the worksheet, using the SL table and his knowledge of the software project. Refer to paragraph 2.3.1 for the method of completing this step, as well as an example.

For each column in the table, the user locates the description that best fits the consideration for that column, then writes the corresponding SL number in the correct row under SL on the worksheet. At the end of this process, the SL column of the worksheet will contain ten values.

Next, the user weights the ten considerations by entering values in the *Weight* column. If all considerations are weighted equally, a *1* is entered in each row of the *Weight* column. If a consideration is critical, a higher weighting (e.g., *3*) is assigned. Weight assignment is subjective and dependent on the project manager's knowledge of the proposed project, but is valuable in emphasizing certain considerations. As discussed in the Path 1 description, the project manager will probably feel more comfortable assigning a *1* for the weighting value, but as he gains experience in using the tables, he will acquire a *feel* for how weighting influences final methodology selection.

The Product column is completed on a row-by-row basis by multiplying the SL by the Weight. For example if the SL is *1* and the weight is *2* the product is 1 X 2 = 2. The *2* is entered in the Product column. Each row on the worksheet is processed similarly, resulting in entries for all considerations in all three columns.

The sum of the weight column is calculated and entered in the *Sum* row; the sum of the product column is calculated and entered in the *Sum* row. The Overall Significance Level (OSL) for the project is calculated as follows:

OSL = Product Sum / Weighting Sum

Thus, if the Product Sum = 11, and the Weighting Sum = 9, the OSL = 1.22. Since whole numbers are required, the fraction must be rounded up or down. To determine this, the consideration is located with the highest weighting and its corresponding SL becomes the determinant. (In the case where two or more considerations are weighted with the same value, then use the highest of their corresponding SL's as the determinant.) If the SL is 2 or greater, the OSL is rounded up; if 1 or less, it is rounded down. For example, if *cost* has the highest weighting and the SL for cost is 1, we round down. The final OSL, in this case, is rounded down from 1.20 to 1 and is entered on the

worksheet.

2.4.2 Step 2 -- Select the Software Category

In Step 2 the user determines which software category best fits his proposed software project. To accomplish this he uses the Software Categories Table, figure 2-6, and enters the result on the Methodology Selection Worksheet. This step is identical to Step 2 in the Path 1 description, paragraph 2.3.2, and the example will not be repeated here.

Using the Software Categories Table, the user reviews the 18 software categories, using his knowledge of the proposed software project, and selects the most relevant category. The software category number is entered in the *Software Category* row of the worksheet.

2.4.3 Step 3 -- Designate Candidate methodologies

In Step 3 the user matches candidate methodologies against desired design methodology capabilities; i.e., the user knows which capabilities he'd like in a methodology (from a design-phase viewpoint) and this step allows him to select those methodologies that come closest to having those capabilities. To accomplish this, the user will need the Path 2 Match Table, figure 2-15, and the Methodology Selection Worksheet, figure 2-3.

A candidate methodology *will have approximately the same pattern of entries* (where *x* entries indicate the capabilities present in a methodology) in a row of the *Methodology* section that the *Software Category* section has in the software category row. The key letters for the candidates are entered on the worksheet.

For instance (see figure 2-16, Example Use of Path 2 Match Table), in software category *6*, entries indicate structuring techniques for all subcolumns under the *Decomposition* and *Abstraction* columns (i.e., the *functional*, *data*, and *control*, (under Decomposition) and *data*, and *process* (under Abstraction) columns contain *x's*).

In the Methodology section, two methodologies have *almost* identical entry patterns, K and L. Thus, methodologies K and L become candidates for final selection and are so noted on the worksheet. Perfect matches may not exist between a software category's capabilities and the methodology capabilities. For example, the B methodology is close to matching and could be selected as a valid candidate, unless its absence of the control decomposition structure would critically affect the project. The user must look for reasonable approximations in selecting candidate methodologies. He may select a set of candidates having either less or more structuring techniques than those desired.

In general, completing Step 3 means the user has selected several candidate methodolo-

Path 2 Match Table						
		Structuring Techniques				
		Decomposition			Abstraction	
		functional	data	control	data	process
SOFTWARE CATEGORY	1		x		x	
	2	x		x		x
	3	x		x		x
	4	x				x
	5	x				
	6	x	x	x	x	x
	7	x		x	x	x
	8	x	x		x	x
	9		x		x	
	10	x		x	x	x
	11	x	x	x	x	x
	12		x		x	
	13	x		x	x	x
	14		x		x	
	15	x				x
	16	x	x		x	
	17	x	x		x	x
	18	x	x		x	
METHODOLOGY	A	x	x	x		
	B				x	x
	C	x	x	x		
	D		x	x		
	E	x	x		x	x
	F	x	x	x		
	G	x	x		x	
	H	x	x		x	
	I	x	x		x	x
	J					x
	K		x	x	x	x
	L	x	x	x	x	x

Figure 2-15 Path 2 Match Table

Path 2 Match Table						
		Structuring Techniques				
		Decomposition			Abstraction	
		functional	data	control	data	process
SOFTWARE CATEGORY	1		x		x	
	2	x		x		x
	3	x		x		x
	4	x				x
	5	x				
	6	x	x	x	x	x
	7	x		x	x	x
	8	x	x		x	x
	9		x		x	
	10	x		x	x	x
	11	x	x	x	x	x
	12		x		x	
	13	x		x	x	x
	14		x		x	
	15	x				x
	16	x	x		x	
	17	x	x		x	x
	18	x	x		x	
METHODOLOGY	A	x	x	x		
	B				x	x
	C	x	x	x		
	D		x	x		
	E	x	x		x	x
	F	x	x	x		
	G	x	x		x	
	H	x	x		x	
	I	x	x		x	x
	J					x
	K	x	x	x	x	x
	L	x	x	x	x	x

software category

matches "L"

Figure 2-16 Example Use of Path 2 Match Table

gies having capabilities critical to the project's life cycle phase and software category.

2.4.4 Step 4 -- Compare Scores for Candidate Methodologies

In Step 4 the user scores each candidate methodology and determines the final methodology for his proposed software project. To accomplish this, he will need the Methodology Selection Worksheet and four figures.

He begins by finding the proper methodology score table for his Path and OSL. On Path 2, he will use figure 2-17 if the OSL is *0*, figure 2-18 if the OSL is *1*, figure 2-19 if the OSL is *2*, and figure 2-20 if the OSL is *3*.

After accessing the correct table for his OSL, he locates the methodology letter under the *Methodology* column, then follows that row to the column under the *Software Category* number. The intersection of the methodology row and the software category column contains the pre-calculated score for the candidate methodology. This score is entered in the *Score* row, under the candidate methodology key letter, on the worksheet.

The candidate methodology that best fits the proposed software project requirements will be the one with a final score closest to zero. That is, if methodology D has a final score of -35 and methodology F has a final score of -32, then methodology F is the best fit.

2.5 METHODOLOGY SELECTION - Path 3

Path 3 differs radically from Paths 1 and 2 which use a project's software category and significance to key tables that list capabilities and lead to eventual methodology selection. Path 3 provides a single table for directly selecting capabilities wanted in a methodology. Although shorter, Path 3 should be used only when a project manager has the experience to stipulate the specific capabilities he wants in a the final methodology. It requires greater project knowledge on the part of the user, and should not be used as a shortcut method by the less experienced project manager. A Path 3 overview is shown in figure 2-21.

This Path is used when the project manager knows which capabilities are of overriding project concern. As stated above, Path 3 ignores significance levels, software categories, and does not require calculation of a score to select methodologies. Instead, the project manager examines a table which lists ratings for capabilities and selects methodologies based on the strength of those ratings.

Path 2
Methodology Scores (OSL = 0)

Methodology	Software Category																	
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
A	20	21	21	19	19	24	21	22	20	21	24	20	21	20	16	23	22	21
B	33	34	34	37	34	37	40	37	33	40	37	33	40	33	25	36	37	34
C	23	23	23	20	20	26	23	23	23	23	26	23	23	23	16	23	23	23
D	28	28	28	26	26	30	28	28	28	28	30	28	28	28	22	31	28	27
E	27	27	27	27	25	31	30	31	27	30	31	27	30	27	19	30	31	27
F	40	39	39	40	40	40	41	39	40	41	40	40	41	40	29	42	39	38
G	22	19	19	20	20	24	23	24	22	23	24	22	23	22	13	26	24	22
H	42	40	40	41	41	45	44	45	42	44	45	42	44	42	27	47	45	42
I	24	21	21	21	18	26	24	26	24	24	26	24	24	24	17	25	26	22
J	27	33	33	33	30	33	33	33	27	33	33	27	33	27	19	30	33	27
K	35	40	40	38	37	43	41	41	35	41	43	35	41	35	27	40	41	37
L	38	35	35	38	35	40	41	38	38	41	40	38	41	38	26	39	38	36

Figure 2-17 Path 2 Methodology Scores for OSL=0

Path 2
Methodology Scores (OSL = 1)

Methodology	Software Category																	
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
A	1	1	1	-1	0	2	-1	1	1	-1	2	1	-1	1	1	1	1	2
B	14	14	14	17	15	15	18	16	14	18	15	14	18	14	10	14	16	15
C	4	3	3	0	1	4	1	2	4	1	4	4	1	4	1	1	2	4
D	9	8	8	6	7	8	6	7	9	6	8	9	6	9	7	9	7	8
E	8	7	7	7	6	9	8	10	8	8	9	8	8	8	4	8	10	8
F	21	19	19	20	21	18	19	18	21	19	18	21	19	21	14	20	18	19
G	3	-1	-1	0	1	2	1	3	3	1	2	3	1	3	-2	4	3	3
H	23	20	20	21	22	23	22	24	23	22	23	23	22	23	12	25	24	23
I	5	1	1	1	1	4	2	5	5	2	4	5	2	5	2	3	5	3
J	8	13	13	13	11	11	11	12	8	11	11	8	11	8	4	8	12	8
K	16	20	20	18	18	21	19	20	16	19	21	16	19	16	12	18	20	18
L	19	15	15	18	16	18	19	17	19	19	18	19	19	19	11	17	17	17

Figure 2-18 Path 2 Methodology Scores for OSL=1

Path 2
Methodology Scores (OSL = 2)

Methodology	Software Category																	
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
A	-18	-19	-19	-21	-19	-20	-23	-20	-18	-23	-20	-18	-23	-18	-14	-21	-20	-17
B	-5	-6	-6	-3	-4	-7	-4	-5	-5	-4	-7	-5	-4	-5	-5	-8	-5	-4
C	-15	-17	-17	-20	-18	-18	-21	-19	-15	-21	-18	-15	-21	-15	-14	-21	-19	-15
D	-10	-12	-12	-14	-12	-14	-16	-14	-10	-16	-14	-10	-16	-10	-8	-13	-14	-11
E	-11	-13	-13	-13	-13	-13	-14	-11	-11	-14	-13	-11	-14	-11	-11	-14	-11	-11
F	2	-1	-1	0	2	-4	-3	-3	2	-3	-4	2	-3	2	-1	-2	-3	0
G	-16	-21	-21	-20	-18	-20	-21	-18	-16	-21	-20	-16	-21	-16	-17	-18	-18	-16
H	4	0	0	1	3	1	0	3	4	0	1	4	0	4	-3	3	3	4
I	-14	-19	-19	-19	-20	-18	-20	-16	-14	-20	-18	-14	-20	-14	-13	-19	-16	-16
J	-11	-7	-7	-7	-8	-11	-11	-9	-11	-11	-11	-11	-11	-11	-11	-14	-9	-11
K	-3	0	0	-2	-1	-1	-3	-1	-3	-3	-1	-3	-3	-3	-3	-4	-1	-1
L	0	-5	-5	-2	-3	-4	-3	-4	0	-3	-4	0	-3	0	-4	-5	-4	-2

Figure 2-19 Path 2 Methodology Scores for OSL=2

Path 2
Methodology Scores (OSL = 3)

Methodology	Software Category																	
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
A	-37	-39	-39	-41	-38	-42	-45	-41	-37	-45	-42	-37	-45	-37	-29	-43	-41	-36
B	-24	-26	-26	-23	-23	-29	-26	-26	-24	-26	-29	-24	-26	-24	-20	-30	-26	-23
C	-34	-37	-37	-40	-37	-40	-43	-40	-34	-43	-40	-34	-43	-34	-29	-43	-40	-34
D	-29	-32	-32	-34	-31	-36	-38	-35	-29	-38	-36	-29	-38	-29	-23	-35	-35	-30
E	-30	-33	-33	-33	-32	-35	-36	-32	-30	-36	-35	-30	-36	-30	-26	-36	-32	-30
F	-17	-21	-21	-20	-17	-26	-25	-24	-17	-25	-26	-17	-25	-17	-16	-24	-24	-19
G	-35	-41	-41	-40	-37	-42	-43	-39	-35	-43	-42	-35	-43	-35	-32	-40	-39	-35
H	-15	-20	-20	-19	-16	-21	-22	-18	-15	-22	-21	-15	-22	-15	-18	-19	-18	-15
I	-33	-39	-39	-39	-39	-40	-42	-37	-33	-42	-40	-33	-42	-33	-28	-41	-37	-35
J	-30	-27	-27	-27	-27	-33	-33	-30	-30	-33	-33	-30	-33	-30	-26	-36	-30	-30
K	-22	-20	-20	-22	-20	-23	-25	-22	-22	-25	-23	-22	-25	-22	-18	-26	-22	-20
L	-19	-25	-25	-22	-22	-26	-25	-25	-19	-25	-26	-19	-25	-19	-19	-27	-25	-21

Figure 2-20 Path 2 Methodology Scores for OSL=3

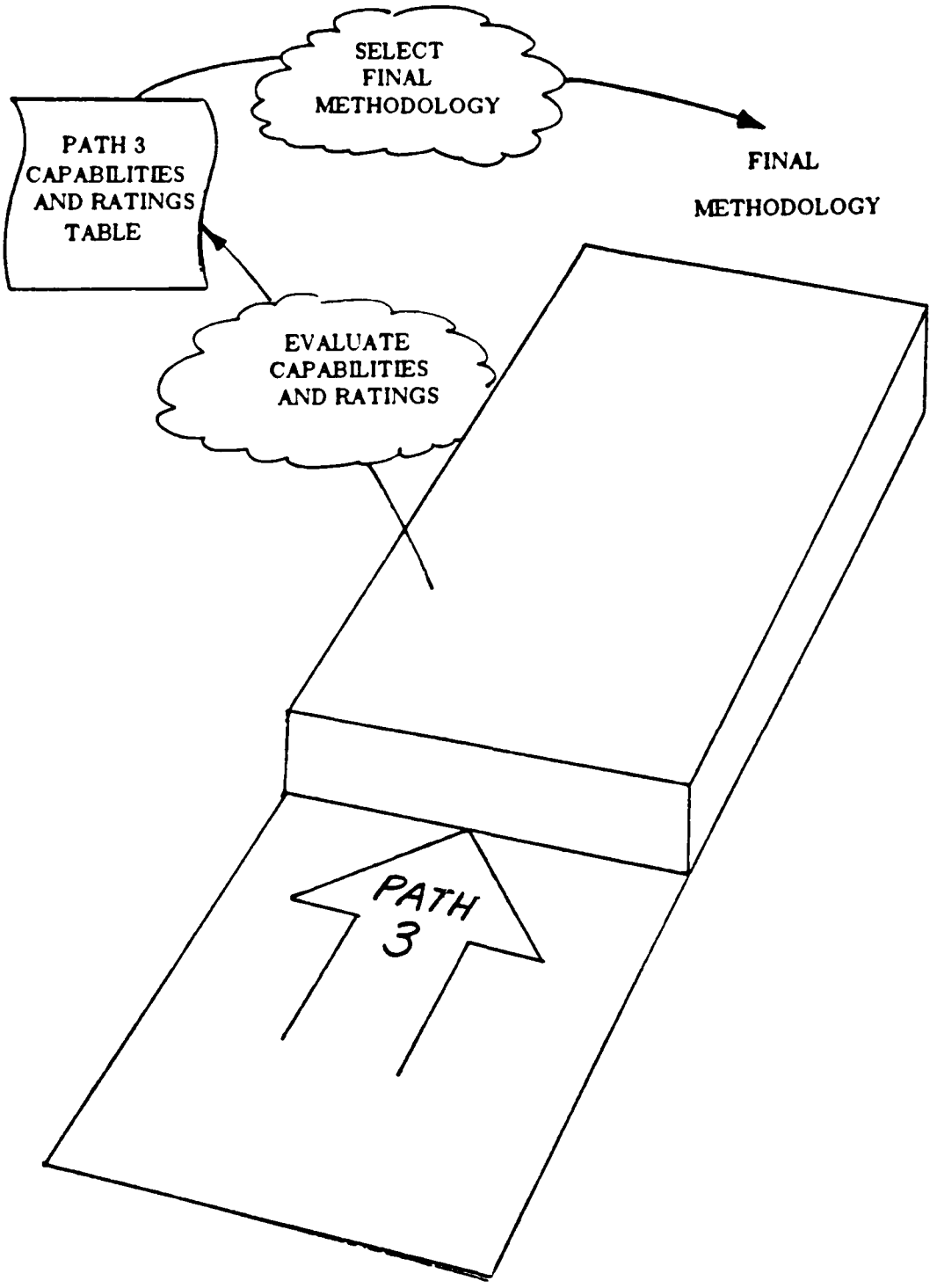


Figure 2-21 Path 3 Overview

Figure 2-22 lists the capabilities and ratings, where applicable, for methodologies A through L. The table is divided into three sections: requirements, design, and universal. The ratings range from 0 to 3; *0* is the least effective rating, *3* is the most effective rating.

In following Path 3, the user locates the capabilities of overriding concern and finds the methodologies which most effectively incorporate those capabilities. He may start with any of the three sections, depending on his areas of greatest concern. A simple way to select among candidate methodologies is to sum each one's ratings and select (among) the one(s) with the highest score(s). There are numerous ways to select methodologies using the Methodology Ratings Table. They can be as simplistic as comparing the scores of one capability or as complicated as comparing methodologies relative to a user-defined fictional-ideal methodology, where the user chooses the individual capabilities and their levels of support.

2.6 OVERALL CONSIDERATIONS

It is important to remember that the first two approaches (i.e., Path 1 and Path 2) for determining the best-fit methodology for a proposed software project represent an attempt to quantify and use three diverse areas of knowledge: (1) the Air Force missions usage and expectations of software development projects, (2) our assessment of specification methodologies available to the software developer, and (3) the project manager's knowledge of what he wants in a methodology specification for his proposed software project. The tables were constructed from samplings taken from the mission surveys and to the extent the surveys represent the missions' software usages and requirements, the results should be accurate. However, evaluation of the final methodology selection should include a careful review of the characteristics of that methodology, found in section 4.0. To find out the names of the candidate methodologies, refer to the Methodology List Table (figure 2-23).

Likewise, Path 3 is even more dependent on the project manager's knowing which capabilities would be most useful during the project's life cycle. This Path is highly subjective and is not recommended for use by an inexperienced project manager. It does, however, provide an expeditious way to select a methodology. If the project manager wanted to verify his choice, he could follow Paths 1 or 2 and compare the results with his Path 3 selection.

It is the final responsibility of the user to ensure that a selected methodology is feasible. Management, availability, hardware restrictions, or other considerations may make the chosen methodology less feasible than another candidate. For example, automated tools for a methodology may require a particular CPU to which the project manager has no access. In this case, selection of that methodology is a useless exercise. The user must exercise common sense in limiting his candidate choices to methodologies that fit within

TABLE OF METHODOLOGY RATINGS

Capability	Methodology											
	A	B	C	D	E	F	G	H	I	J	K	L
REQUIREMENTS												
state modeling	0	1	0	2	1	3	1	1	1	1	0	2
data flow modeling	3	0	2	3	0	1	0	0	1	0	2	2
control flow modeling	2	0	3	2	0	1	0	0	0	0	2	2
object modeling	1	3	0	1	3	1	3	3	3	3	3	0
timing performance spec	0	0	1	1	1	2	0	0	0	3	3	0
accuracy performance spec	0	2	1	0	1	2	2	2	0	3	0	0
DESIGN												
functional decomposition	2	0	3	0	2	1	1	2	2	0	0	2
data decomposition	3	0	3	2	1	1	2	2	2	0	2	1
control decomposition	2	0	3	2	0	1	0	0	0	0	2	1
data abstraction	0	3	0	0	3	0	3	3	3	0	1	3
process abstraction	0	3	0	0	2	0	0	0	3	3	1	2
data base definition	1	2	0	3	1	3	2	2	2	0	0	3
concurrency/synchronicity	1	0	0	1	2	1	2	3	1	3	3	0
module interface definition	1	3	0	1	1	2	2	2	1	3	3	3
formal verification	0	3	0	0	0	2	1	1	0	0	0	3
configuration management	0	0	0	0	0	0	0	0	0	0	0	2
completeness analysis	1	3	1	1	1	3	1	3	1	3	3	3
consistency analysis	1	3	1	1	1	3	1	3	1	3	3	3
Ada compatibility	1	2	2	1	3	2	1	3	1	3	2	2
code behavior notation	0	3	0	3	2	3	3	3	2	3	0	3
UNIVERSAL												
prototyping	0	0	0	1	0	2	0	2	0	3	0	2
test plan generation	0	0	0	0	0	0	0	3	0	0	2	0
automated tool available	1	3	3	3	0	3	0	3	0	3	3	3
traceability	0	3	1	1	1	3	1	3	1	0	3	0
transistion between phases	1	3	1	2	1	3	1	3	1	0	3	2
validation	2	3	1	1	2	3	1	3	2	3	3	2
usability	2	1	1	2	3	1	1	1	2	1	1	2
maturity	3	3	3	3	2	3	2	1	2	2	2	2
training/experience level	2	1	2	2	3	3	1	1	3	1	2	1
MIL-STD documentation	1	0	1	1	2	2	0	1	0	0	2	1

Figure 2-22 Path 3 Capabilities and Ratings

METHODOLOGY and TOOL LIST

Key	Acronymn	Name
Mature Methodologies		
A	DSSD	Data Structured Systems Design
B	HDM	Hierarchical Development Method
C	SADT	Structured Analysis and Design Technique
		tools:
C_a		*TAGS*
D	SA/SD	Structured Analysis and Structured Design (Realtime Yourdon)
		tools:
D_a		*ARGUS*
D_b		*EXCELERATOR*
D_c		*PROMOD*
E	SCR	Software Cost Reduction - Navy
F	SREM	Software Requirements Engineering Methodology
G	VDM	Vienna Development Method
Evolving Methodologies		
H	DCDS	Distributed Computing Design System
I	JSD	Jackson System Design
J	PAISLey	Process-oriented, Applicative, Interpretable Specification Language
K	SARA	System ARchitect's Apprentice
L	USE	User Software Engineering Methodology

Figure 2-23 Methodology List Table

his project's environmental constraints.

2.7 EXAMPLE USE OF GUIDELINES – C^{3}I SYSTEM

The first part of this section provides the guidelines for selecting a methodology and supporting tools for a project, by software category and with full consideration given to life cycle phase.

The second part of this section is an example of how the guidelines and tables are used -- in this case, for a C^{3}I system.

The objective is to present several C^{3}I examples using the three paths previously defined. A primary consideration imposed on each example will be compatibility with the Ada programming language. Of course, a project manager uses more than a single consideration in determining which methodology best fits his future project. In deriving the following C^{3}I system requirements and design considerations, reference was made to the C^{3}I mission appendix, Appendix C, actual requirements set forth in C^{3}I RFP's, and working knowledge of the requirements for C^{3}I software and system projects gained by Boeing Aerospace engineers during the last decade.

2.7.1 Path 1 Example C^{3}I System

The example depicts development of C^{3}I system software for interactive displays. The example system is ground-based and receives data off-line that has been collected by airborne sensors. The overall system requirements are for a prototype system that will process data collected by airborne sensors and classify types of ground threats and their locations. Software requirements include: the ability to control and respond to cursor location on display, the ability to access and change an associated data base, user friendliness, and use of a high-order language, such as Ada. Additional requirements are for a good data base management system and the compatibility of data products during all phases of the software development life cycle.

The four steps for determining the proper requirements methodology are as follows:

2.7.1.1 Step 1 -- Choose the Overall Significance Level (OSL)

First, the user examines the Significance Level Table, figure 2-2, and fills out the Methodology Selection Worksheet, figure 2-3. Note that for this example, the Significance Level Table was examined and the following decisions made:

1. The *Cost* consideration was *Some Cost Flexibility* and rated a significance level (SL) of *2*.

2. The *Criticality* consideration was *Nuisance Impact* and rated an SL of *1*.

3. The *Schedule* consideration was *Normal Schedule Constraints* and rated an SL of *2*.

4. The *Complexity* consideration was *Moderate Complexity* and rated an SL of *1*.

5. The *Development Formality* consideration was *Normal to Strong Contractor Controls* and rated an SL of *1*.

6. The *Software Utility* consideration was *Prototype* and rated an SL of *0*.

7. The *Reliability* consideration was *Respond correctly to nominal conditions* and rated an SL of *0*.

8. The *Correctness* consideration was *functionality and constraints met* and rated an SL of *1*.

9. The *Maintainability* consideration was *predict impact of changes* and rated an SL of *1*.

10. The *Verifiability* consideration was *Full complement of documentation; design documentation updated too* and rated an SL of *2*.

These significance level values were entered on the worksheet under the *SL* column. Figure 2-24 is a completed worksheet showing the above SL values.

The ten considerations were weighted. Having no particular information as to the critical nature of the considerations, a normal weighting of *1* was assigned to all considerations, except for maintainability, which was weighted with a *3* (since the project manager might justly conclude that maintainability of the completed project cannot be overemphasized).

The SL and weight values were multiplied on a row-by-row basis to fill in the *Product* column.

The *Weight* and *Product* columns were summed. The weight sum was *12* and the product sum was *13*.

The Overall Significance Level (OSL) for the project was the product sum divided by the weight sum. The result of this division was *1.08* and needed to be rounded up or down to a whole number. The rounding rule, listed in paragraph 2.3.1, is two-part: (1)

METHODOLOGY SELECTION WORKSHEET

SOFTWARE TO BE ACQUIRED *Interactive Display C³I System*
LIFECYCLE PHASE ..... x REQUIREMENTS ___ DESIGN

CONSIDERATIONS	SL (0, 1, 2, 3) FIGURE 2-2	WEIGHT (1 = NORMAL)	PRODUCT (WEIGHT X SL)
COST	2	1	2
CRITICALITY	1	1	1
SCHEDULE	2	1	2
COMPLEXITY	1	1	1
DEVELOPMENT FORMALITY	1	1	1
SOFTWARE UTILITY	0	1	0
RELIABILITY	0	1	0
CORRECTNESS	1	1	1
MAINTAINABILITY	1	3	3
VERIFIABILITY	2	1	2
SUM	[redacted]	12	13
OSL = SUM OF PRODUCT / SUM OF WEIGHT = $\frac{13}{12}$ = 1.08 round down → 1			
SOFTWARE CATEGORY			14
CANDIDATE METHODOLOGIES			
KEY	F	C	
SCORE	22	4	

Figure 2-24 C³I Path 1 Example
Use of Methodology Selection Worksheet

the consideration is found with the highest weighting and its corresponding SL value noted, (2) if the SL is *2* or greater, the OSL is rounded up; if *1* or less, it is rounded down.

In the example, the consideration having the highest weighting (of *3*) was *MAINTAINABILITY*. The corresponding SL was *1*. Since the rule is to round down when the SL of the highest weight (for a consideration) is *1* or less, the OSL was changed from *1.08* to *1*. Note that if two or more considerations qualified as having the same high weighting, we would have picked the one(s) with the highest SL as the rounding determinant.

2.7.1.2 Step 2 -- Select the Best-Fit Software Category

The Software Categories Table, figure 2-6, is examined for the software category that best fits the nature of the project.

Software Category 14, Data presentation, was selected as the best fit. The number *14* was recorded on the worksheet in the *Software Category* row.

2.7.1.3 Step 3 -- Designate Candidate Methodologies

Referring to the Path 1 Match Table, figure 2-7, we examine row *14* in the *Software Category* section. We note that row *14* contains two *x's* corresponding to modeling techniques and one *x* corresponding to accuracy specification. In the *Methodology* section of the table, we search for rows that also contain *x's* for the same columns. Examination reveals that the *F* methodology matches exactly (i.e., it has an *x* in every column required by software category *14* -- plus three additional *x*'s). Thus, an *F* is entered on the worksheet as a candidate in the *Key* row. Methodologies *A, B, C, D, E, G, H, I, J, and K* contain *x*'s for two of the columns. For the sake of brevity of the example, it is decided that data flow modeling and accuracy specification are the most important capabilities to provide. Of the methodologies listed above, only *C* provides capabilities for both data flow modeling and accuracy specification. Thus, a *C* is entered on the worksheet as another candidate methodology. Figure 2-25 shows the Path 1 Match Table with the above rows circled to illustrate Step 3.

2.7.1.4 Step 4 -- Compare Scores for Candidate Methodologies

As discussed in paragraph 2.3.4, Step 4 requires the user to locate the proper methodology score table that corresponds to the OSL of the project (the OSL is on the worksheet). The OSL for the C^3I example is *1* and figure 2-10, Path 1 Methodology Scores for OSL of 1, is used. A C^3I Path 1 example use of this table is shown in figure 2-26.

Path 1 Match Table								
		Modeling Techniques				Performance Specification		
		State	Flow		Object	Timing	Accuracy	
			Data	Control				
SOFTWARE CATEGORY	1		x		x		x	
	2			x		x		
	3			x		x		
	4		x	x		x		
	5		x	x	x	x	x	
	6	x	x	x	x	x		
	7	x	x	x	x	x	x	
	8		x	x	x	x		
	9		x		x		x	
	10	x	x	x	x	x	x	
	11	x	x	x	x	x		
	12		x		x		x	
	13	x	x	x	x	x	x	
	14		x		x		x	
	15	x	x	x	x		x	
	16		x	x	x		x	
	17		x	x	x	x		
	18		x		x			
METHODOLOGY	A		x	x	x			
	B	x			x		x	
	C		x	x		x	x	
	D	x	x	x	x	x		
	E	x			x	x	x	
	F	x	x	x	x	x	x	
	G	x			x		x	
	H	x			x		x	
	I	x	x		x			
	J	x			x	x	x	
	K		x	x	x	x		
	L	x	x	x				

Figure 2-25 C^{3}I Path 1 Example
Use of Match Table

Path 1
Methodology Scores (OSL = 1)

Find Software Category

Methodology	Software Category																	
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
A	2	1	1	1	1	3	-1	3	2	-1	3	2	-1	2	2	4	3	4
B	16	12	12	14	15	14	18	15	16	18	14	16	18	16	11	16	15	16
C	4	5	5	3	3	5	2	4	4	2	5	4	2	4	2	4	4	4
D	10	9	9	9	9	12	9	10	10	9	12	10	9	10	10	12	10	10
E	9	6	6	5	6	9	8	10	9	8	9	9	8	9	14	9	10	9
F	22	20	20	21	23	21	23	19	22	23	21	22	23	22	17	22	19	19
G	5	-3	-3	-3	1	1	1	2	5	1	1	5	1	5	-1	6	2	4
H	25	18	18	18	22	22	22	23	25	22	22	25	22	25	13	27	23	24
I	6	-1	-1	-1	-2	4	1	5	6	1	4	6	1	6	2	4	5	5
J	11	14	14	13	15	13	15	14	11	15	13	11	15	11	6	11	14	9
K	18	23	23	22	23	26	23	26	18	23	26	18	23	18	14	22	26	21
L	18	15	15	19	15	19	19	17	18	19	19	18	19	18	12	18	17	17

Find Methodology Key Letters

Figure 2-26 C^{3}I Path 1 Example
Use of Methodology Scores for OSL=1

The user finds rows *C* and *F* and follows these across to column *14*. The intersections of these two rows and the column contain the scores for the two methodologies, when used for a project of software category *14*, provided the OSL is *1*.

The final selection is made based on which key letter's score is closest to zero. In this example the scores are:

1. Methodology C score = 4

2. Methodology F score = 22

The methodology whose score is closest to zero and that best fits the C^3I project is represented by the key letter C. To find out the requirements methodology selected, the user refers to Methodology List Table, figure 2-23. Note that the methodology selected is the Structured Analysis and Design Technique (SADT) methodology.

The final step is to read the SADT methodology description in section 4.0 and ensure that it generally meets overall project environmental considerations.

Remember that additional capabilities listed at the beginning of this example included compatibility of life-cycle products and compatibility with Ada. In reviewing the description of SADT, we find the following:

1. SADT supports decomposition very well. The diagrams record both data and control flow information.

2. SADT has no basic incompatibility with Ada, although the diagrams it produces do not directly map to Ada features.

3. TAGS and STRADIS are independent tool sets that support the SADT methodology.

4. SADT is mature.

5. A manager can learn SADT techniques in less than a month, while a developer may need from one to three months to fully understand all its applications.

From the above, the user may assume that SADT is moderately user friendly (doesn't take long to learn, that it supports life-cycle phases (through use of graphics products), that it is readily available and that it has supporting tool sets. Therefore, SADT can be considered mature and low-risk.

Should the project manager be dissatisfied with the final selection, he may examine the descriptions on his other candidate methodologies for one more compatible with his project requirements. His other alternatives are to try the Path 2 approach (i.e., select the

candidate methodologies using design considerations), or the Path 3 approach (i.e., select the candidate methodologies on the basis of those capabilities that are of overriding project concern).

The point is, no set of tables can adequately address all the variables needed to select methodologies for all future software projects and be correct every time. This document contains guidelines to aid the project manager in making his selection, but he must be the final judge of that selection.

2.7.2 Path 2 Example -- C^3I System

The Path 2 example approaches methodology selection from the viewpoint of design capabilities. The example is for an airborne software system for use on an unnamed special purpose computer, but for which the software can be developed, in Ada, using a standard Air Force CPU. Further, the C^3I software must be interactive with remote terminals, user friendly, incorporate a special-purpose data base, and support graphics in near-real time.

The four steps for determining the proper requirements methodology are as follows:

2.7.2.1 Step 1 -- Choose the Overall Significance Level (OSL)

First, the user examined the Significance Level Table, figure 2-2, and filled out the Methodology Selection Worksheet, figure 2-3. Note that for this example, the Significance Level Table was examined and the following decisions made:

1. The *Cost* consideration was *Normal cost constraints* and rated a significance level (SL) of *1*.

2. The *Criticality* consideration was *Mission Impact* and rated an SL of *2*.

3. The *Schedule* consideration was *Normal Schedule Constraints* and rated an SL of *2*.

4. The *Complexity* consideration was *Greater Complexity* and rated an SL of *2*.

5. The *Development Formality* consideration was *Strong contractual controls; Formal reviews* and rated an SL of *2*.

6. The *Software Utility* consideration was *Real-time, Avionics, C^3I software* and rated an SL of *2*.

7. The *Reliability* consideration was *Faults removed ASAP* and rated an SL of *2.*

8. The *Correctness* consideration was *implementation validated against the design specification* and rated an SL of *2.*

9. The *Maintainability* consideration was *Extent of changes optimally localized* and rated an SL of *3.*

10. The *Verifiability* consideration was *Requirements through source code documentation always up-to-date* and rated an SL of *3.*

These significance level values were entered on the worksheet under the *SL* column. Figure 2-27 is a completed worksheet showing the above SL values.

The ten considerations were then weighted. Having no particular information as to the critical nature of the considerations, all ten were weighted normally, i.e., with a *1.*

The SL and weight values were multiplied on a row-by-row basis to fill in the *Product* column.

The *Weight* and *Product* columns were summed. The weight sum was *10* and the product sum was *26.*

The Overall Significance Level (OSL) for the project was the product sum divided by the weight sum. The result of this division was *2.6* and needed to be rounded up or down to a whole number. The rounding rule, listed in paragraph 2.3.1, was two-part: (1) the consideration with the highest weighting was found and its corresponding SL value noted, (2) if the SL was *2* or greater, the OSL was rounded up; if *1* or less, it was rounded down.

In the example, all the weightings were normal (i.e., *1*) and thus all qualified as being "highest" weightings. When two or more weightings qualify as being highest, we check the corresponding SL's for their highest values. Thus, in this case, we find that two considerations have SL's of *3* (maintainability and verifiability). Since the SL used as the rounding determinant was *2* or more, rounding was up and the OSL changed from *2.6* to *3.*

2.7.2.2 Step 2 -- Select the Best-Fit Software Category

The user examined the Software Categories Table, figure 2-6, for the software category that best fit the nature of his project.

METHODOLOGY SELECTION WORKSHEET

SOFTWARE TO BE ACQUIRED Realtime C^3I Software
LIFECYCLE PHASE .....___REQUIREMENTS _x_DESIGN

CONSIDERATIONS	SL (0, 1, 2, 3) FIGURE 2-2	WEIGHT (1 = NORMAL)	PRODUCT (WEIGHT X SL)
COST	1	1	1
CRITICALITY	2	1	2
SCHEDULE	2	1	2
COMPLEXITY	2	1	2
DEVELOPMENT FORMALITY	2	1	2
SOFTWARE UTILITY	2	1	2
RELIABILITY	2	1	2
CORRECTNESS	2	1	2
MAINTAINABILITY	3	1	3
VERIFIABILITY	3	1	3
SUM	[redacted]	10	26
OSL = SUM OF PRODUCT / SUM OF WEIGHT = $\frac{26}{10} = 2.6$ round up → 3			
SOFTWARE CATEGORY			2

CANDIDATE METHODOLOGIES				
KEY	E	I	K	L
SCORE	-33	-39	-20	-25

Figure 2-27 C^3I Path 2 Example
Use of Methodology Selection Worksheet

Software Category 2, Event Control, was selected as the best fit. The number *2* was recorded on the worksheet in the *Software Category* row.

2.7.2.3 Step 3 -- Designate Candidate Methodologies

The user referred to the Path 2 Match Table, figure 2-15, and examined row *2* in the *Software Category* section. He noted that row *2* contained three *x's* corresponding to structuring techniques. He looked for rows in the *Methodology* section of the table that contained *x's* for the same columns. His examination revealed that the *L* methodology contained the same structuring techniques needed by the category 2 software, along with additional techniques. No other methodology contained all three of the category 2 software techniques. At least two of the structuring techniques desired were supplied by methodologies *A, C, D, E, F, I, and K.* Since Ada was designed to use the concept of abstractions and since one project consideration is Ada compatibility, the other candidate methodologies selected from the list above were those that supported the process abstraction technique. The candidates then became *E, I, K, and L.* Figure 2-28 shows the Path 2 Match Table with the above rows circled to illustrate Step 3. These key letters were entered on the worksheet under the *Candidate Methodologies* section in the *Key* row.

2.7.2.4 Step 4 -- Compare Scores for Candidate Methodologies

As discussed in paragraph 2.3.4, Step 4 required the user to locate the proper methodology score table that corresponds to the OSL of the project (the OSL is on the worksheet). The OSL for the C^3I example was *3* and figure 2-20, Path 2 Methodology Scores for OSL of 3, was used.

The user found rows *E, I, K,* and *L* and followed these across to column *2*, the appropriate software category. The intersections of these four rows and the column contained the scores for the four methodologies (when used for a project of software category *2*, provided the OSL was *3*).

The final selection was made based on which key letter's score was closest to zero. Figure 2-29 provides an example of the use of this table. In this example the scores were:

1. Methodology E score = -33
2. Methodology I score = -39
3. Methodology K score = -20

Path 2 Match Table						
		Structuring Techniques				
		Decomposition			Abstraction	
		functional	data	control	data	process
SOFTWARE CATEGORY	1		x		x	
	2	x		x		x
	3	x		x		x
	4	x				x
	5	x				
	6	x	x	x	x	x
	7	x		x	x	x
	8	x	x		x	x
	9		x		x	
	10	x		x	x	x
	11	x	x	x	x	x
	12		x		x	
	13	x		x	x	x
	14		x		x	
	15	x				x
	16	x	x		x	
	17	x	x		x	x
	18	x	x		x	
METHODOLOGY	A	x	x	x		
	B				x	x
	C	x	x	x		
	D		x	x		
	E	x	x		x	x
	F	x	x	x		
	G	x	x		x	
	H	x	x		x	
	I	x	x		x	x
	J					x
	K		x	x	x	x
	L	x	x	x	x	x

Figure 2-28 C^{3}I Path 2 Example
Use of Match Table

Path 2
Methodology Scores (OSL = 3)

Methodology	Software Category																	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
A	-37	-39	-39	-41	-38	-42	-45	-41	-37	-45	-42	-37	-45	-37	-29	-43	-41	-36
B	-24	-26	-26	-23	-23	-29	-26	-26	-24	-26	-29	-24	-26	-24	-20	-30	-26	-23
C	-34	-37	-37	-40	-37	-40	-43	-40	-34	-43	-40	-34	-43	-34	-29	-43	-40	-34
D	-29	-32	-32	-34	-31	-36	-38	-35	-29	-38	-36	-29	-38	-29	-23	-35	-35	-30
E	-30	-33	-33	-33	-32	-35	-36	-32	-30	-36	-35	-30	-36	-30	-26	-36	-32	-30
F	-17	-21	-21	-20	-17	-26	-25	-24	-17	-25	-26	-17	-25	-17	-16	-24	-24	-19
G	-35	-41	-41	-40	-37	-42	-43	-39	-35	-43	-42	-35	-43	-35	-32	-40	-39	-35
H	-15	-20	-20	-19	-16	-21	-22	-18	-15	-22	-21	-15	-22	-15	-18	-19	-18	-15
I	-33	-39	-39	-39	-39	-40	-42	-37	-33	-42	-40	-33	-42	-33	-28	-41	-37	-35
J	-30	-36	-36	-37	-37	-33	-33	-30	-30	-33	-33	-30	-33	-30	-26	-36	-30	-30
K	-22	-20	-20	-22	-20	-23	-25	-22	-22	-25	-23	-22	-25	-22	-18	-26	-22	-20
L	-19	-25	-25	-22	-22	-26	-25	-25	-19	-25	-26	-19	-25	-19	-19	-27	-25	-21

Figure 2-29 C³I Path 2 Example
Use of Methodology Scores for OSL=3

4. Methodology L score = -25

Of the four methodologies, the scores for *K* and *L* were closest to zero. Those two methodologies were developed and exercised in an academic environment and needed to be closely inspected by the project manager to ensure they adequately addressed his project requirements.

The methodology whose score was closest to zero and that best fit the C^3I project was represented by the key letter K. To find out which requirements methodology was selected, the user referred to Methodology List Table, figure 2-23. Note that the methodology selected was the System ARchitect's Apprentice (SARA) methodology.

The final step involved looking up the SARA methodology description in section 4.0 and ensuring that it generally met overall project environmental considerations.

Remember that additional capabilities listed at the beginning of this example included: data base management, user friendliness, compatibility of life-cycle products, and compatibility with Ada. If, in reviewing the description of SARA, the project manager was dissatisfied with it, he had the same recourse described in the Path 1 C^3I example, namely that of reading the descriptions for methodologies *E, I,* and *L*; and determining whether one of them was a better fit for his project. His other alternatives were to try the Path 1 approach (i.e., selecting the candidate methodologies using requirements considerations), or the Path 3 approach (i.e., selecting the candidate methodologies on the basis of those capabilities that were of overriding project concern).

As discussed at the end of the Path 1 C^3I example, no set of tables can adequately address all the variables needed to select methodologies for all future software projects and be correct every time. This document contains guidelines to aid the project manager in making his selection, but he must be the final judge of that selection.

2.7.3 Path 3 Example -- C^3I System

Path 3 is provided for the project manager who has learned from experience those capabilities he wants in a methodology. Whatever the project, he approaches it from one of three viewpoints: requirements, design, or universal capabilities.

For example, suppose he felt the methodology must be strongest in the requirements capabilities. He turned to figure 2-22 and examined the ***Requirements*** section of the table. Inspection revealed that the *F* methodology was best and, he noted, its design and universal capabilities were strong. Figure 2-30, C^3I Path 3 Example Use of Methodology Ratings Table, shows the selection process.

TABLE OF METHODOLOGY RATINGS

Capability	Methodology											
	A	B	C	D	E	F	G	H	I	J	K	L
REQUIREMENTS												
state modeling	0	1	0	2	1	3	1	1	1	1	0	2
data flow modeling	3	0	2	3	0	1	0	0	1	0	2	2
control flow modeling	2	0	3	2	0	1	0	0	0	0	2	2
object modeling	1	3	0	1	3	1	3	3	3	3	3	0
timing performance spec	0	0	1	1	1	2	0	0	0	3	3	0
accuracy performance spec	0	2	1	0	1	2	2	2	0	3	0	0
DESIGN						10		6				6
functional decomposition	2	0	3	0	2	1	1	2	2	0	0	2
data decomposition	3	0	3	2	1	1	2	2	2	0	2	1
control decomposition	2	0	3	2	0	1	0	0	0	0	2	1
data abstraction	0	3	0	0	3	0	3	3	3	0	1	3
process abstraction	0	3	0	0	2	0	0	0	3	3	1	2
data base definition	1	2	0	3	1	3	2	2	2	0	0	3
concurrency/synchronicity	1	0	0	1	2	1	2	3	1	3	3	0
module interface definition	1	3	0	1	1	2	2	2	1	3	3	3
formal verification	0	3	0	0	0	2	1	1	0	0	0	3
configuration management	0	0	0	0	0	0	0	0	0	0	0	2
completeness analysis	1	3	1	1	1	3	1	3	1	3	3	3
consistency analysis	1	3	1	1	1	3	1	3	1	3	3	3
Ada compatibility	1	2	2	1	3	2	1	3	1	2	2	2
code behavior notation	0	3	0	3	2	3	3	3	2	3	0	3
UNIVERSAL						22		29				31
prototyping	0	0	0	1	0	2	0	2	0	3	0	2
test plan generation	0	0	0	0	0	0	0	3	0	0	2	0
automated tool available	1	3	3	3	0	3	0	3	0	3	3	3
traceability	0	3	1	1	1	3	1	3	1	0	3	0
transistion between phases	1	3	1	2	1	3	1	3	1	0	3	2
validation	2	3	1	1	2	3	1	3	2	3	3	2
usability	2	1	1	2	3	1	1	1	2	1	1	2
maturity	3	3	3	3	2	3	2	1	2	2	2	2
training/experience level	2	1	2	2	3	3	1	1	3	1	2	1
MIL-STD documentation	1	0	1	1	2	2	0	1	0	0	2	1
						23		21				15

F = 10 + 22 + 23 = 55
H = 6 + 29 + 21 = 54
L = 6 + 31 + 15 = 52

Figure 2-30 C^{3}I Path 3 Example
Use of Methodology Ratings Table

Suppose the project manager felt the design life-cycle phase was the most important, so he looked for the methodology with the most strength in design capabilities. He selected the *L* methodology and noted that it was weaker in the requirements area, but moderately strong in the universal capabilities area.

Finally, suppose he decided that the universal capabilities were most important for his project and chose two methodologies, *F* and *H*. Methodology *F* was already chosen as the best candidate in the requirements capabilities area; methodology *H* was nearly as strong in the universal area, stronger than *F* in the design area, but weak in the requirements phase.

The project manager may have examined capabilities, checking in all three areas (requirements, design, universal) or in only one area. He may have summed scores of selected capabilities for the three methodologies and selected the highest. If he did this for all the capabilities, he found that the sums were: $F = 55$, $H = 54$, and $L = 52$. A spread of three points wasn't much. His final resort was to refer to the Methodology List Table, figure 2-23, where he found the names of his candidates, then read their descriptions in section 4.0.

In the previous examples for Paths 1 and 2, we discussed how this document contains guidelines to aid the project manager in making his selection, but that he must be the final judge of that selection. This Path 3 example is no different. It is a short-cut technique for the experienced manager to use in finding a methodology for his project. Again, the final decision is his and must be based on his experience in project management.

2.7.4 Blank Worksheets

The following blank Methodology Selection Worksheets are included as "tear-outs" for use by project managers during methodology selection. If all copies of the worksheet have been removed, figure 2-3 can be removed from this guidebook, reproduced, and returned to the guidebook for the next user.

METHODOLOGY SELECTION WORKSHEET

SOFTWARE TO BE ACQUIRED _____

LIFECYCLE PHASE ..... ___REQUIREMENTS ___DESIGN

CONSIDERATIONS	SL (0, 1, 2, 3) FIGURE 2-2	WEIGHT (1 = NORMAL)	PRODUCT (WEIGHT X SL)
COST			
CRITICALITY			
SCHEDULE			
COMPLEXITY			
DEVELOPMENT FORMALITY			
SOFTWARE UTILITY			
RELIABILITY			
CORRECTNESS			
MAINTAINABILITY			
VERIFIABILITY			
SUM			
OSL = SUM OF PRODUCT / SUM OF WEIGHT =			
SOFTWARE CATEGORY			
CANDIDATE METHODOLOGIES			
KEY			
SCORE			

METHODOLOGY SELECTION WORKSHEET

SOFTWARE TO BE ACQUIRED _____
LIFECYCLE PHASE ..... ___REQUIREMENTS ___DESIGN

CONSIDERATIONS	SL (0, 1, 2, 3) FIGURE 2-2	WEIGHT (1 = NORMAL)	PRODUCT (WEIGHT X SL)
COST			
CRITICALITY			
SCHEDULE			
COMPLEXITY			
DEVELOPMENT FORMALITY			
SOFTWARE UTILITY			
RELIABILITY			
CORRECTNESS			
MAINTAINABILITY			
VERIFIABILITY			
SUM			
OSL = SUM OF PRODUCT / SUM OF WEIGHT =			
SOFTWARE CATEGORY			
CANDIDATE METHODOLOGIES			
KEY			
SCORE			

METHODOLOGY SELECTION WORKSHEET

SOFTWARE TO BE ACQUIRED _____
LIFECYCLE PHASE ..... ___REQUIREMENTS ___DESIGN

CONSIDERATIONS	SL (0, 1, 2, 3) FIGURE 2-2	WEIGHT (1 = NORMAL)	PRODUCT (WEIGHT X SL)
COST			
CRITICALITY			
SCHEDULE			
COMPLEXITY			
DEVELOPMENT FORMALITY			
SOFTWARE UTILITY			
RELIABILITY			
CORRECTNESS			
MAINTAINABILITY			
VERIFIABILITY			
SUM			
OSL = SUM OF PRODUCT / SUM OF WEIGHT =			
SOFTWARE CATEGORY			
CANDIDATE METHODOLOGIES			
KEY			
SCORE			

METHODOLOGY SELECTION WORKSHEET

SOFTWARE TO BE ACQUIRED _____

LIFECYCLE PHASE ..... ___REQUIREMENTS ___DESIGN

CONSIDERATIONS	SL (0, 1, 2, 3) FIGURE 2-2	WEIGHT (1 = NORMAL)	PRODUCT (WEIGHT X SL)
COST			
CRITICALITY			
SCHEDULE			
COMPLEXITY			
DEVELOPMENT FORMALITY			
SOFTWARE UTILITY			
RELIABILITY			
CORRECTNESS			
MAINTAINABILITY			
VERIFIABILITY			
SUM			
OSL = SUM OF PRODUCT / SUM OF WEIGHT =			
SOFTWARE CATEGORY			
CANDIDATE METHODOLOGIES			
KEY			
SCORE			

METHODOLOGY SELECTION WORKSHEET

SOFTWARE TO BE ACQUIRED _____
LIFECYCLE PHASE .....____REQUIREMENTS ____DESIGN

CONSIDERATIONS	SL (0, 1, 2, 3) FIGURE 2-2	WEIGHT (1 = NORMAL)	PRODUCT (WEIGHT X SL)
COST			
CRITICALITY			
SCHEDULE			
COMPLEXITY			
DEVELOPMENT FORMALITY			
SOFTWARE UTILITY			
RELIABILITY			
CORRECTNESS			
MAINTAINABILITY			
VERIFIABILITY			
SUM			
OSL = SUM OF PRODUCT / SUM OF WEIGHT =			
SOFTWARE CATEGORY			
CANDIDATE METHODOLOGIES			
KEY			
SCORE			

METHODOLOGY SELECTION WORKSHEET

SOFTWARE TO BE ACQUIRED _____

LIFECYCLE PHASE .....___REQUIREMENTS ___DESIGN

CONSIDERATIONS	SL (0, 1, 2, 3) FIGURE 2-2	WEIGHT (1 = NORMAL)	PRODUCT (WEIGHT X SL)
COST			
CRITICALITY			
SCHEDULE			
COMPLEXITY			
DEVELOPMENT FORMALITY			
SOFTWARE UTILITY			
RELIABILITY			
CORRECTNESS			
MAINTAINABILITY			
VERIFIABILITY			
SUM			
OSL = SUM OF PRODUCT / SUM OF WEIGHT =			
SOFTWARE CATEGORY			
CANDIDATE METHODOLOGIES			
KEY			
SCORE			

METHODOLOGY SELECTION WORKSHEET

SOFTWARE TO BE ACQUIRED _____

LIFECYCLE PHASE ..... ___REQUIREMENTS ___DESIGN

CONSIDERATIONS	SL (0, 1, 2, 3) FIGURE 2-2	WEIGHT (1 = NORMAL)	PRODUCT (WEIGHT X SL)
COST			
CRITICALITY			
SCHEDULE			
COMPLEXITY			
DEVELOPMENT FORMALITY			
SOFTWARE UTILITY			
RELIABILITY			
CORRECTNESS			
MAINTAINABILITY			
VERIFIABILITY			
SUM			
OSL = SUM OF PRODUCT / SUM OF WEIGHT =			
SOFTWARE CATEGORY			
CANDIDATE METHODOLOGIES			
KEY			
SCORE			

METHODOLOGY SELECTION WORKSHEET

SOFTWARE TO BE ACQUIRED _____
LIFECYCLE PHASE ..... ___REQUIREMENTS ___DESIGN

CONSIDERATIONS	SL (0, 1, 2, 3) FIGURE 2-2	WEIGHT (1 = NORMAL)	PRODUCT (WEIGHT X SL)
COST			
CRITICALITY			
SCHEDULE			
COMPLEXITY			
DEVELOPMENT FORMALITY			
SOFTWARE UTILITY			
RELIABILITY			
CORRECTNESS			
MAINTAINABILITY			
VERIFIABILITY			
SUM			
OSL = SUM OF PRODUCT / SUM OF WEIGHT =			
SOFTWARE CATEGORY			
CANDIDATE METHODOLOGIES			
KEY			
SCORE			

METHODOLOGY SELECTION WORKSHEET

SOFTWARE TO BE ACQUIRED _____
LIFECYCLE PHASE ..... ___REQUIREMENTS ___DESIGN

CONSIDERATIONS	SL (0, 1, 2, 3) FIGURE 2-2	WEIGHT (1 = NORMAL)	PRODUCT (WEIGHT X SL)
COST			
CRITICALITY			
SCHEDULE			
COMPLEXITY			
DEVELOPMENT FORMALITY			
SOFTWARE UTILITY			
RELIABILITY			
CORRECTNESS			
MAINTAINABILITY			
VERIFIABILITY			
SUM			
OSL = SUM OF PRODUCT / SUM OF WEIGHT =			
SOFTWARE CATEGORY			
CANDIDATE METHODOLOGIES			
KEY			
SCORE			

3.0 HOW TO SELECT AVAILABLE AUTOMATED TOOLS

3.1 INTRODUCTION

Automated tools can provide invaluable assistance during the requirements and design phases of the life cycle. They can assure consistency of identifiers and terms, enforce both documentation and project standards, and enhance tracking of project progress. The tools can also free the project manager from tedious, repetitive tasks.

The choice of selecting automated tools is trivial if a tool set was developed hand-in-hand with the methodology. That situation exists for the DSSD, HDM, SREM and DCDS, PAISLey, SARA, and USE methodologies.

The choice of selecting automated tools may require comparing several alternative tool sets developed for use with a particular methodology. That situation exists for SADT and SA/SD (Yourdon Methodology). Section 3.3 explains how to compare alternative tool sets.

The choice of selecting automated tools from available generic tools becomes necessary when no specific tool set supports a methodology. Generic tools support various software engineering tasks. Section 3.4 discusses the selection of generic tools.

3.2 THE SELECTION PROCESS

The process of selecting tools is guided by a set of questions that determine which of the three situations described above is pertinent. Figure 3-1 is a schematic representation of the process. The process is set forth in the following text.

Question 1: Does a tool set exist that is integral to the project's intended methodology?

If the methodology is either DSSD, HDM, SREM or DCDS, PAISLey, SARA, or USE, the answer is YES, and the tool set is described in section 4.0. Proceed to question 3.

If the tool set is not integral to the methodology, the answer is NO, proceed to question 2.

Schematic of Selection Process

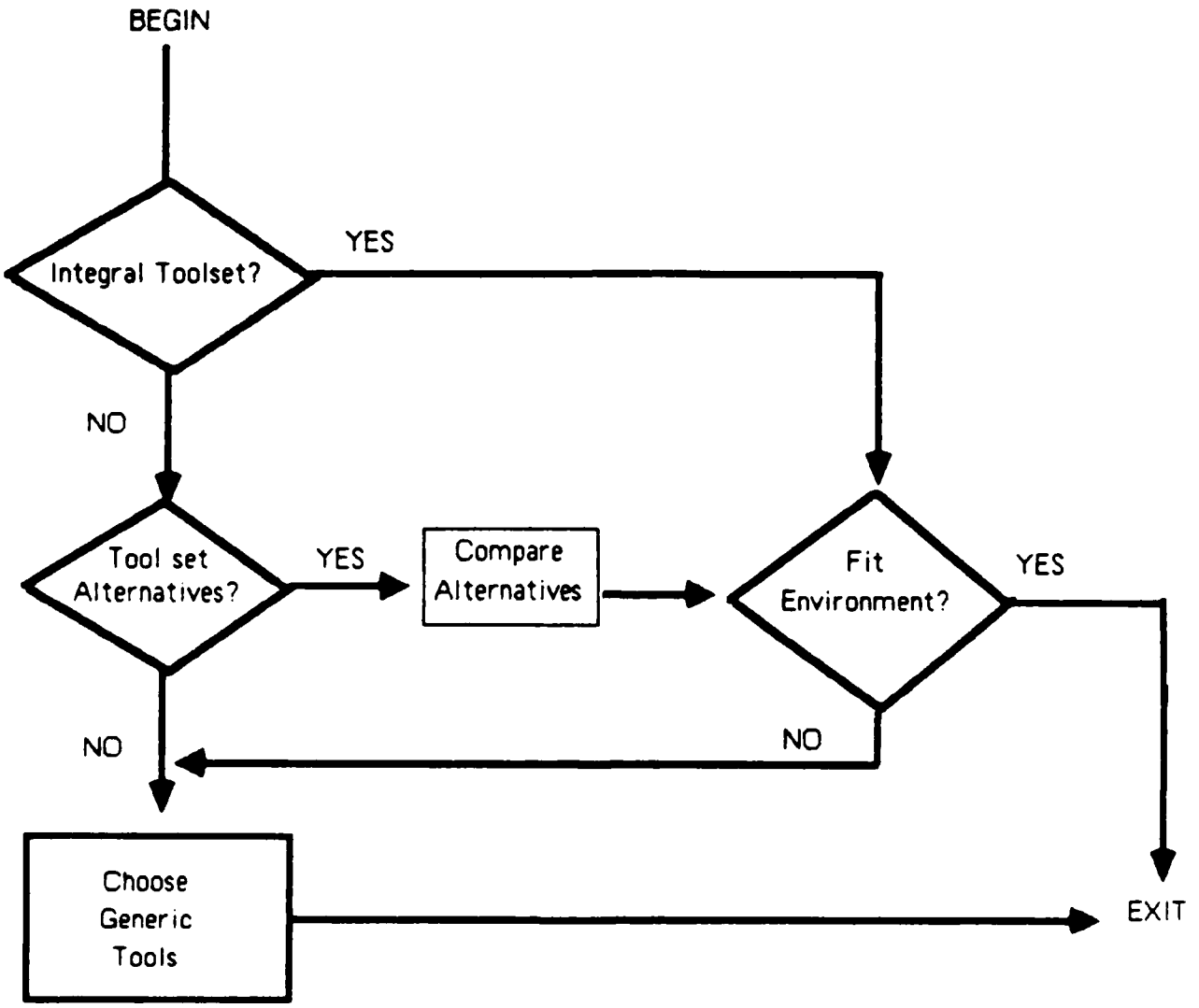


Figure 3-1 Tool Selection Process

Question 2: Do tool set alternatives exist that support the project's intended methodology?

If the methodology is either SADT or SA/SD (Yourdon Methodology), the answer is YES. Use the comparison and selection process described in section 3.3, and then proceed to question 3.

If no tool set alternatives exist, use the selection process for generic tools described in section 3.4. (Note that project environment usability is tested by various criteria in the selection process for generic tools.)

Question 3: Are the tools usable in the project environment?

Typical considerations involved in answering question 3 are:

— Is the specified CPU or hardware readily obtainable?

— Are the costs involved (time and money) reasonable for the project?

— If the specifications are to be developed by a team, do the tools allow and control sharing of data among the team members?

If the answer to question 3 is YES, then the tool selection process is complete.

If the answer is NO, use the selection process for generic tools described in section 3.4.

3.3 COMPARISON AND SELECTION PROCESS FOR TOOL SET ALTERNATIVES

The considerations involved in selecting one set of tools over another reflect the following:

- Cost and schedule considerations:
  - What is the cost of the software license for the tool set?
  - What will training cost in terms of time and money?
  - What will be the hardware costs associated with the tool set?
  - Is the hardware needed readily available or obtainable?

- Software considerations:
  - Does the complexity of the project and its development formality warrant automated support?
- Quality considerations:
  - Which is most important: reliability, correctness, maintainability, or verifiability?
  - How do the tool set alternatives compare in supporting that most important consideration?
  - How do the tool set alternatives compare in supporting the other quality considerations?

The evaluations in section 4.0 are limited to four tool set alternatives: (1) TAGS, (2) ARGUS, (3) EXCELERATOR, and (4) PROMOD, whose descriptions can be found in sections 4.16, 4.17, 4.18, and 4.19 respectively. When a selected methodology is associated with one of these tool sets, the corresponding description should be analyzed in terms of the most important considerations of the project. In the case where the project manager wants to evaluate other tool sets (e.g., tool sets that have become available since publication of this guidebook, or that have been enhanced with new capabilities, etc.), he may find it useful to produce results in the standardized format found in section 4.15. This will facilitate comparing his new evaluations against those in the guidebook.

In addition to commercially available tool sets, some companies have developed tools for their own use (e.g., STRADIS). Such tools are not usually available to other users, but may be considered as a viable alternative tool set when that company is being evaluated by the Air Force for a particular project.

The alternatives for the SADT methodology are TAGS and tool sets developed by large companies for their own use (STRADIS is an example). The alternatives for SA/SD are ARGUS, EXCELERATOR, PROMOD and Hughes has a tool set based on the Yourdon methodology.

Thus, it is a good idea to review the current promotional literature on a particular tool set as part of the selection process.

3.4 SELECTION PROCESS FOR GENERIC TOOLS

The purpose of selecting a set of generic tools is to create an automated environment that is of direct benefit to the personnel developing the specifications. If the tools are difficult to use or to coordinate, then their potential benefits are reduced. It is therefore important that a project manager fully evaluate a potential set of tools. One approach to evaluating tools (as mentioned in section 3.3), is to follow the format in section 4.15 of this guidebook to assure that relevant questions are asked about the tool set.

As with methodology selection, the significance of the project should fit the level of support the tools provide. Although some form of automated document preparation or control is always useful, less significant projects need less support than more significant projects.

Figure 3-2 lists generic tools that could be useful in an automated environment for specification development.

A well-known example of a generic tool that is versatile enough to have been used on many projects is PSL/PSA. It combines the function of a Data Flow Checker with some of the functions of a Project Database Management System. An evaluation of PSL/PSA can be found in section 4.20.

An example of a generic tool is BYRON, recently released by Intermetrics. It is a PDL language system, intended for use with and in an Ada environment, but has not been available long enough to be considered "mature." This newness is one of the factors that must be evaluated when selecting a generic tool.

Figure 3-3 lists criteria for rating individual tools. These criteria provide a checklist of considerations that should be reviewed when selecting generic tools. There are many generic tools available, and a listing and evaluation of them is beyond the scope of this guidebook.

GENERIC TECHNIQUE	DESCRIPTION	ADVANTAGE	DISADVANTAGE
Project Database Management System	A database and tools used to manage the results (documents, specifications) of a project.	Provides positive visibility of project status, convenient query and reports of specifications, and access control.	Requires resources for establishing and maintaining database. Complicates developer and system interface.
Data Flow Checker	Tests completeness and consistency of defined producers and consumers of data, the flow of data between producers and consumers, and the relationships between data elements.	Assists specification validation process and long-term maintenance of documentation.	Requires training in preparation of input and comprehension of results.
Formal specification language	The use of a formal, computer-processable notation for expressing specifications.	Specifications can be described by effect rather than procedure. Specifications can be statically and dynamically analysed for completeness and consistency.	Extensive training in formal logic and mathematics and experience necessary.
Interactive Graphics	Use of computers to produce, maintain, and change such two-dimensional graphical images as data flow diagrams and structure charts.	Easy to modify.	Requires relatively larger amounts of computer overhead to contain graphics software.
PDL Language System	The use of a stylised or formal notation to express a detailed design.	Design can be completed and verified before coding begins. Assists partitioning of design into units to be coded by a single individual. Enhances communication between designer and programmer	Tempts designer into coding an implementation rather than creating a design.

Figure 3-2 GENERIC SPECIFICATION TOOLS (part 1 of 2)

GENERIC TECHNIQUE	DESCRIPTION	ADVANTAGE	DISADVANTAGE
Simulation or prototyping	A model, used to predict performance, check or demonstrate functionality, determine impact of change, or obtain information on system capacity.	Most effective technique for studying transient behavior.	Relatively expensive and time-consuming to develop accurate model(s).
Static Analyzer	Detection of errors through examination of specifications written in a formal notation. Errors that can be detected are: syntax, misspellings, missing statements, and improper sequencing of statements.	Cost of error detections is low.	Effectiveness is limited by the known properties that can be checked.
Specification Tree	Automated generation of a display of the hierarchical relationships within a related set of documents.	Provides a layered roadmap of documents.	May ignore more important relationships.
Traceability Analyzer	A systematic search to ensure that the contents of two results or documents have the claimed relationships.	Assists in verifying software development products meet requirements (e.g., verifying design against requirements).	Tedious for developer to establish relationships.
Word Processing	The use of a computer to store a document in a digitized form so that it may be edited, manipulated, and stored.	Easily supports changes to documents.	Requires investment in hardware and training. There is a lack of interchange standards.

Figure 3-2 GENERIC SPECIFICATION TOOLS (part 2 of 2)

Criterion	Definition
Technical Feasibility	An assessment of the technical problems of integrating a tool into the existing environment. Considerations include unusual data requirements or overly large processing or storage demands.
Payoff/Contribution	An assessment of relative usefulness of a tool.
Estimated Cost	An assessment of cost which includes acquisition, integration, and usage costs.
Life-cycle data	An assessment of the specification life cycles in which the tools are most useful.
Data Base Compatibility	An assessment of the difficulty in transferring data from this tool to other tools already chosen.
Usable Output	An assessment of the value of the outputs to a specifier or other tools.
Side Effects	An assessment of positive or negative side effects produced. For instance, a tool may generate secondary output that greatly simplifies the tasks of other tools; or a tool may impose inordinate timing, data, storage, or format burden.
Costs/Schedule	An assessment of cost/schedule effects during the life cycle as a result of incorporating the tool into the environment.
Management Benefits	An assessment of the management benefits resulting from use of the tool.
Required Training	An assessment of the cost and duration of specialized training needed before using the tool.

Figure 3-3 RATING CRITERIA FOR GENERIC TOOLS (part 1 of 2)

Criterion	Definition
Usage Constraints	An assessment of the dependency of the tool on specific hardware and software.
Required Computer Resources	An assessment of the processing and storage requirements of a tool.
Level of Human Interaction	An assessment of the level of human interaction (volume of data entered, number of keystokes, etc.) required for use of the tool.
User Interface	An assessment of the user-friendliness of a tool and the compatibility of its interface with other tools in the environment.
Support of Modern Practice	An assessment of the extent to which a tool supports modern specification practice.
Support of Unique Project Needs	An assessment of the applicability or extensibility of the tool to support unique project requirements not addressed by other tools.
Source Code Availability	An assessment of the availability of a tool's source code for evaluation and/or modification.

Figure 3-3 RATING CRITERIA FOR GENERIC TOOLS (part 2 of 2)

4.0 Methodology and Automated Tool Descriptions

4.1 Organization of this Section

This section contains descriptions of methodologies and tool sets in a standard format. An outline of the format used for methodologies with a description of the contents of each item follows as section 4.2. The methodology descriptions are contained in sections 4.3 through 4.14. They are arranged in the same order as they appear in the tables of section 2.0.

A description of the format used to describe tool sets is contained in section 4.15. The individual descriptions are contained in sections 4.16 through 4.19. A description of PSL/PSA, a generic automated tool for requirements analysis, is found in section 4.20.

To make it easier to identify what methodology or tool set description you are reading, you will find its acronym and key on the top of each page. The acronym is centered; the key is to the right. Notice that the key for a tool set consists of an upper case letter followed by an underscore followed by a lower case letter (eg., X_x). The upper case letter identifies what methodology the tool set supports. The lower case letter differentiates the tool sets themselves. So, the tool set key of *D_b* informs you that the tool set supports methodology D and there is at least one alternative to consider.

4.2 Methodology Description Format

1. **General Aspects**

   A. Identification

   Gives the name and acronym of the methodology and identifies the developing/supporting organization.

   B. Overview

   Contains a short description of the salient features of the methodology.

   C. Identifies the specification life cycle phases supported:

   Requirements Analysis, Architectural Design (intermodule communication, data structures), or Detailed Design (module functionality).

   Complementary methodologies will be listed for phases not supported.

D. Software Categories

Lists standard software categories which are compatible with this methodology.

#	Category
1	Arithmetic-based
2	Event Control
3	Process Control
4	Procedure Control
5	Navigation
6	Flight Dynamics
7	Orbital Dynamics
8	Message Processing
9	Diagnostic S/W
10	Sensor/signal Processing
11	Simulation
12	Database Management
13	Data acquisition
14	Decision/planning aids
15	Data presentation
16	Pattern/image processing
17	Computer System Software
18	S/W development tools

E. Suitable for systems of size:

— Small (<2,000 lines of code)

— Medium (2,000 - 10,000 lines of code)

— Large (>10,000 lines of code)

2. Technical Aspects

A. Primary approach

For a requirements methodology, the approaches are:

— flow-oriented,

— object-oriented, and

— state-oriented.

For a design methodology, the approaches are:

— data-structured,

— decomposition,

— encapsulation, and

— programming calculus.

B. Supports

Traceability
Functional hierarchy/decomposition
Data hierarchy/data abstraction
Interface definition
Database definition
Data flow
Sequential control flow
Concurrency/parallelism
Formal program verification
Iterative development

C. Workproducts

Are they relevant to MIL-STD documentation?

a. Textual

Descriptions of reports, documents produced.

b. Graphical

Descriptions of diagrams produced.

D. Performance Specification

Does the methodology have the capability to specify or test timing and/or accuracy constraints that apply to individual system functions?

E. Operating Qualities Specification

Does the methodology have the capability to specify the following constraints?

— Man/machine interaction

— Fault-tolerance

— Portability

— Reusability

— Security

F. Ada compatibility

Ada Feature	Supported
Packages	*X*
Tasks	
Generics	
Exception Handling	*C*
Types	
Representations	

X indicates support of feature.
C indicates conflict with feature.

G. Quality Assurance

How does the methodology check or enforce:

— Consistency ?

— Completeness ?

— Validation ?

H. Independent of

Are the resulting specifications independent of:

— Implementation Language ?

— Hardware Architecture ?

— Operating System Architecture ?

3. **Support Aspects**

A. Automated Tools

Describes which automated tools are available.

B. Language

Identifies the language used in the following specification phases and its degree of formality.

— Requirements Specification

— Architectural Design

— Detailed Design

4. Management Aspects

Does the methodology support project, technical, or configuration management? How?

5. Usage Aspects

A. Equipment/Facilities Needed to use

Identify specific hardware and software (operating systems, graphics packages) required to use the methodology or associated automated tools.

B. Usability

Level	Methodology
Easy to Use	
Moderately Easy to Use	
Moderately Difficult to Use	
Difficult to Use	

C. Extent of Use

Is the methodology mature? Has it been used outside the developing organization? How much?

6. Transferability

A. *Availability*

Is the methodology in the public domain, commercially available, etc.?

B. *Training Available*

— Public documentation

— Proprietary documentation

— Consultants

— Seminars - scheduling and cost, if known

C. *Training and Experience Required*

Training/Experience Needed			
months	USER	MANAGER	ORGANIZATION
< 1			
1 - 3			
3 - 6			
> 6			

The table entries reflect the amount of training and experience time required to use the methodology effectively. A USER is an individual who develops or assists in developing requirements and/or design specifications. An ORGANIZATION is a group of users developing specifications as a team.

D. *Primary Source of Documentation*

List references.

4.3 DSSD Methodology Description

1. General Aspects

A. *Identification*

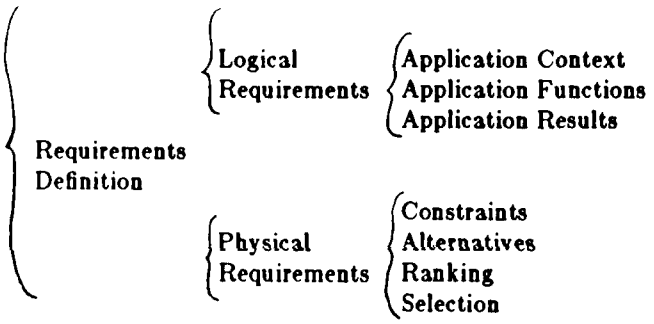
DSSD - Data Structured System Design

Ken Orr & Assoc, Inc
1725 Gage Blvd
Topeka, KS 66604-3379
(913) 233-0653
(800) 255-2459

B. *Overview*

DSSD is a data-structured development methodology. The basic idea is to define outputs and their structure and then to work backwards to inputs.

The basic technique is to construct hierarchically structured diagrams called assembly line diagrams that read left to right instead of top to bottom. The diagrams can be structured to represent a hierarchy of processing steps, events in time, data flow, or data structures. An example of an assembly line diagram which illustrates what constitutes requirements definition as recommended by DSSD is found below.



The methodology uses entity diagrams to model the software system and its environment and to model functional flow. Detailed design for processes (transformations) is done through a variant of Warnier-Orr diagrams in which the process is always found at the leftmost bottom edge of the diagram.

C. *Life cycle phases supported:*

All three phases (Requirements Analysis, Architectural Design, and Detailed Design) are addressed.

D. *Software Categories*

#	category
1	Arithmetic-based
9	Diagnostic Software
12	Database Management
14	Data presentation
17	Computer System Software
18	Software Development tools

E. *Suitable for systems of size:*

Can be used for development of any size system.

2. Technical Aspects

A. *Primary approach*

Dataflow-oriented for requirements; data-structured for design.

B. *Supports*

	Capability
	Traceability
X	Functional hierarchy/decomposition
X	Data hierarchy/data abstraction
X	Interface definition
X	Database definition
X	Data flow
X	Sequential control flow
X	Concurrency/parallelism -*doesn't prohibit*
	Formal program verification
	Iterative development

C. *Workproducts*

Some of the workproducts (assembly line dataflow diagrams, entity diagrams) can be used in preparation of MIL-STD software development documentation.

a. *Textual*

Structured requirements definitions, database design, structured program design

b. *Graphical*

functional flow diagrams
entity diagrams
assembly line diagrams for data flow
input/output diagrams
event structures

D. *Performance Specification*

Performance requirements for specific components is not addressed.

E. *Operating Qualities Specification*

Operating qualities such as man/machine interaction or portability are not addressed.

F. *Ada compatibility*

Ada Feature	Supported
Packages	X
Tasks	X
Generics	C
Exception Handling	X
Types	X
Representations	

G. *Quality Assurance*

Structured walkthroughs provide manual validation of completeness and consistency of requirements and design.

H. *Independence*

The methodology is independent of planned implementation language, hardware architecture, and operating system. However, the tools available assume the implementation language will be COBOL. Tools and an orientation for Ada are under development.

3. Support Aspects

A. *Automated Tools*

The tool set available for DSSD is called STRUCTURE(S). It draws diagrams on a lineprinter and provides a COBOL code generator. STRUCTURE(S) is available for IBM, Honeywell, Univac, and Perkin-Elmer CPUs.

B. *Language*

All languages used for requirements specification and design are graphical. Each language is based on some form of the Warnier-Orr diagram.

4. Management Aspects

The tools provide a project management tool for version control. Technical (quality) management is provided by recommending structured walkthroughs.

5. Usage Aspects

A. *Equipment/Facilities Needed for DSSD use:*

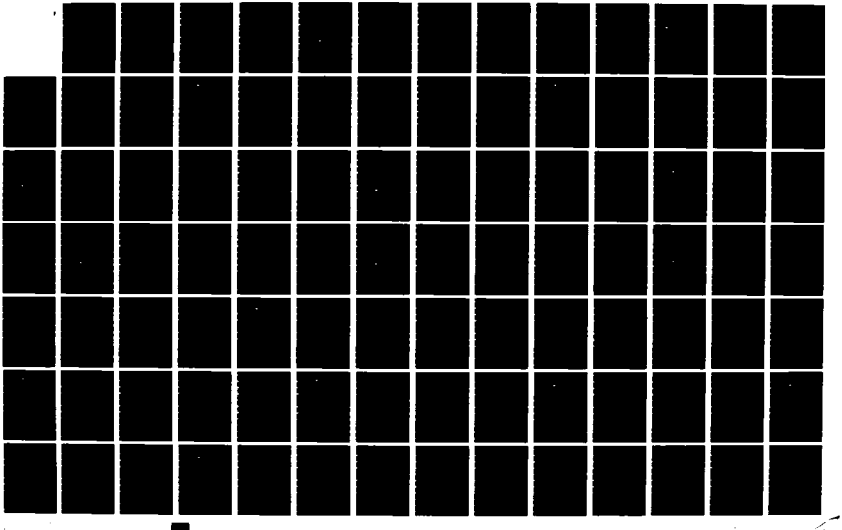
Although diagrams can be produced manually, use of the tools in STRUCTURE(S) requires an IBM, Honeywell, Univac, or Perkin-Elmer CPU and a lineprinter.

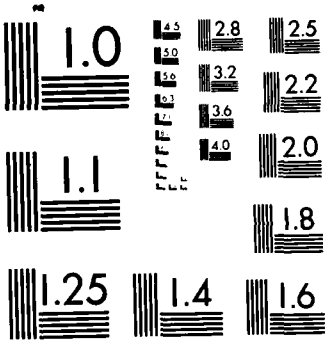
B. *Usability*

Level	Methodology
Easy to Use	
Moderately Easy to Use	X
Moderately Difficult to Use	
Difficult to Use	

AD-A162 457 SPECIFICATION TECHNOLOGY GUIDEBOOK(U) BOEING AEROSPACE 2/3
CO SEATTLE WA D R ADDLEMAN ET AL. AUG 85
RADC-TR-85-135 F30602-84-C-0073

UNCLASSIFIED F/G 9/2 NL





C. *Extent of Use*

The methodology is mature. It has been used by many different organizations in developing information systems projects.

6. Transferability

A. *Availability*

The methodology and tools are commercially available from Ken Orr and Associates.

B. *Training Available*

Ken Orr and Associates provide proprietary documentation, consultants, and privately arranged and regularly scheduled public seminars. The per-person fees for seminars are in the range of $500 to $1000.

C. *Training and Experience Required*

Training/Experience Needed			
months	USER	MANAGER	ORGANIZATION
< 1	X		
1 - 3			
3 - 6			
> 6			X

D. *Primary Source of Documentation*

Ken Orr and Associates

Also see:

References

[Brackett1983].
Michael H. Brackett, *Developing Data Structured Information Systems,* Ken Orr & Associates, Inc, Topeka, Kansas, 1983.

[Orr1977].
Ken Orr & Associates, Inc., *Data Structured Systems Development Methodology,* Ken Orr & Associates, Inc, Topeka, Kansas, 1977.

[Orr1981].
Ken Orr, *Structured Requirements Definition,* Ken Orr & Associates, Inc, Topeka, Kansas, 1981.

4.4 HDM Methodology Evaluation

1. General Aspects

A. Identification

HDM - Hierarchical Development Methodology

Computer Science Laboratory
SRI International
Menlo Park, CA
(415) 859-4771

B. *Overview*

HDM combines the data structured and algorithmic refinement approaches to design. A specification written in HDM is a hierarchy of abstract machines. The methodology assumes that the requirements for the software system have been captured in a model and can be written as a top-level specification known as a TLS.

The TLS describes the system's external (observable) behavior and it is written a nonprocedural language called SPECIAL. Beginning with the TLS, a hierarchy of abstract machines is produced by providing mappings from the representations in the higher level machine to the representations in the lower level machine. Each lower level machine provides more and more concrete detail. Eventually, the lowest level can be translated into an implementation language.

The system was primarily developed for the purpose of verifying security properties of software. The automated tools support formal verification of design. SRI is expecting to make an enhanced HDM available in March 1985, which will be Ada-compatible (concurrency properties planned), through the DoD Security Center.

C. *Life cycle phases supported:*

The two design phases are supported. HDM assumes a requirements specification has already been written.

D. *Software Categories*

#	category
6	Flight Dynamics
7	Orbital Dynamics
8	Message Processing
10	Sensor and Signal Processing
13	Data Acquisition
15	Decision and Planning Aids
16	Pattern and Image Processing

E. *Suitable for systems of size:*

The system could be used to develop systems of any size. However, the use of HDM for a very large system might be unwieldy. In such a case, it would be appropriate to use HDM in developing components for which it is necessary to verify security properties.

2. Technical Aspects

A. *Primary approach*

Combines data-structured and decomposition approaches to design.

B. *Supports*

	Capability
X	Traceability - *(Each level explicitly mapped to the lower level)*
X	Functional hierarchy/decomposition
X	Data hierarchy/data abstraction
X	Interface definition
	Database definition
	Data flow
	Sequential control flow
X	Concurrency/parallelism *(doesn't prohibit, explicitly support in future)*
X	Formal program verification
X	Iterative development

C. *Workproducts*

Not directly relevant to MIL-STD documentation.

a. *Textual*

Produces a series of formal specifications written in SPECIAL which are used for formal verification of the design or implementation.

b. *Graphical*

No graphical workproducts are produced.

D. *Performance Specification*

The specification and measurement of timing constraints are not addressed but accuracy specification is. Indeed, the accuracy originally specified is preserved down the hierarchy of machines.

E. *Operating Qualities Specification*

The security properties desired in the software system to be developed can be derived from its TLS. Further, the presence or absence of those properties is checked for in the design or implementation since either or both can be verified relative to the TLS.

F. *Ada compatibility*

Ada Feature	Supported
Packages	X
Tasks	X
Generics	X
Exception Handling	X
Types	X
Representations	

G. *Quality Assurance*

Consistency and completeness checking are provided by tools which process the language SPECIAL. In addition, specific properties can be validated with the aid of a automated theorem prover.

H. *Independence*

Although the HDM methodology is not intrinsically language dependent, there are tools specifically designed for verifying implementations in PASCAL, MODULA, and Ada (planned for release in 1985).

The security model provided assumes a non-distributed system. If a model of a distributed system was created, then HDM could be used to verify security properties in a distributed environment.

3. Support Aspects

A. *Automated Tools*

HDM includes a set of tools that check the specifications for syntax errors, type errors, consistency, and some aspects of completeness.

B. *Language*

The language SPECIAL (SPECIfication and Assertion Language) is non-procedural, with a formal syntax and semantics. SPECIAL supports modularity, strong typing, user-defined types, exception conditions, assertions, and invariants.

a. *Requirements Specification* - SPECIAL

b. *Architectural Design* - HSL (Hierarchy Specification Language) is used to describe structuring of modules into abstract machines and of machines into systems.

c. *Detailed Design* - SPECIAL

4. Management Aspects

HDM addresses technical management aspects of a software development by providing tools for formal validation of design specifications.

5. Usage Aspects

A. *Equipment/Facilities Needed for HDM use:*

Tools run under TOPS-20 or TENEX operating system and expect INTERLISP.

B. *Usability*

Level	Methodology
Easy to Use	
Moderately Easy to Use	
Moderately Difficult to Use	X
Difficult to Use	

C. *Extent of Use*

HDM is a mature technology. It has been used by various organizations through the auspices of the DoD Security Center.

6. Transferability

A. *Availability*

HDM is available by arrangement with the DoD Security Center. It is installed on CPU's available through the ARPANET.

B. *Training Available*

There is a manual on HDM and SPECIAL available from SRI for approximately $50.

The Mitre Corporation offers public and private seminars on HDM.

C. *Training and Experience Required*

Must be familiar with concepts in logic and formal mathematics.

Training/Experience Needed			
months	USER	MANAGER	ORGANIZATION
< 1			
1 - 3			
3 - 6	X		
> 6			X

D. *Primary Source of Documentation*

SRI International
Mitre Corporation in Bedford, Massachuetts

The reference listed below is an excellent article comparing various formal methodologies for specification and verification of secure operating systems. Its bibliography can be used for locating in-depth articles on this topic.

Also see:

References

[HDM1981].
"Verifying Security," *ACM Computing Surveys*, vol. 13, no. 3, pp. 279-340, Cheheyl, M.H. et al, September 1981.

4.5 SADT Methodology Evaluation

1. General Aspects

A. Identification

SADT - Structured Analysis and Design Technique

Douglas Ross
Softech, Inc.
460 Totten Pond Road
Waltham, MA 02254

B. Overview

SADT is a disciplined decomposition approach to modeling complex problems and systems. The language of SADT (SA) combines a blueprint-like graphics language with the nouns and verbs which describe the problem domain of the system to be modeled.

A SADT diagram, known as an actigram, is composed of no more than six basic blocks. Each basic building block is drawn as a box with four sides called INPUT, CONTROL, OUTPUT, and MECHANISM. Arrows coming in and out of the basic box represent flows of data or control. The box represents a transformation from a *before* state to an *after* state. Thus, an actigram can represent a decomposition of data or activity.

C. Life cycle phases supported:

SADT supports the modeling aspect of requirements analysis very well. However, as a technique for design it is limited to expressing a decomposition.

D. Software Categories

#	category
2	Event Control
3	Process Control
4	Procedure Control
5	Navigation
11	Simulation

E. Suitable for systems of size:

SADT is a suitable methodology for requirements analysis of any size system since it allows the analyst to deal with a limited scope at any one time and the language insures a consistent decomposition.

2. Technical Aspects

A. Primary approach

The SADT approach is one strict decomposition by data or function.

B. Supports

	Capability
X	Traceability - *as a management procedure*
X	Functional hierarchy/decomposition
X	Data hierarchy/data abstraction
X	Interface definition
	Database definition
X	Data flow
X	Sequential control flow
	Concurrency/parallelism
	Formal program verification
X	Iterative development - *by revision of diagrams*

C. Workproducts

The diagrams SADT produces could be used as part of a MIL-STD requirements specification.

The workproducts are all diagrams, no text.

D. Performance Specification

Timing and accuracy constraints can be associated with a diagram, although there is no way to verify that they are consistent with the diagram.

E. Operating Qualities Specification - not addressed.

F. Ada compatibility

SADT could be used for the requirements analysis phase of a software

development project whose implementation language would be Ada since SADT itself is independent of implementation languages. However, SADT does not directly support or map to specific Ada features.

G. Quality Assurance

Handled as a series of procedures for author/reader cycles, structured walk-throughs. Correct use of the diagrams forces some consistency.

H. Independent of:

implementation language, hardware architecture, and operating system architecture.

3. Support Aspects

A. Automated Tools

There is a tool (SCG) available from SofTech that will assist in preparation of the diagrams. TAGS and STRADIS are alternative forms of this methodology. Each has its own tool set which include automated analysis tools.

B. Language

The language is a rigorous graphical language with formal syntax and informal semantics.

a. Requirements Specification - graphical

b. Architectural Design - graphical

c. Detailed Design - could use a PDL

4. Management Aspects

No procedures for project management are included. However, in support of technical management, SADT has procedures for validation of the workproducts, and recommends procedures for checking accuracy and traceability.

5. Usage Aspects

A. Equipment/Facilities Needed for SADT use:

See specific tool set descriptions for TAGS and STRADIS. Also, a diagram construction assistant program is available on a CDC Cybernet or Dec PDP-11 under UNIX version 7.

B. Usability

Level	Methodology
Easy to Use	
Moderately Easy to Use	
Moderately Difficult to Use	X
Difficult to Use	

C. Extent of Use

It has been used in a variety of settings by many organizations.

6. Transferability

A. Availability

In the public domain.

B. Training Available - from various training firms offering courses in Structured System Analysis and design.

C. Training and Experience Required

Training/Experience Needed			
months	USER	MANAGER	ORGANIZATION
< 1		X	
1 - 3	X		
3 - 6			X
> 6			

D. Primary Source of Documentation:

Softech

References

[SADT1977a].
Douglas T. Ross and Kenneth E. Schoman, Jr., "Structured Analysis for Requirements Definition," *IEEE Transactions on Software Engineering*, vol. SE-3, no. 1, pp. 6-15, January 1977.

[SADT1977b].
Douglas T. Ross, "Structured Analysis (SA): A Language for Communicating Ideas," *IEEE Transactions on Software Engineering*, vol. SE-3, no. 1, pp. 16-34, January 1977.

4.6 Yourdon Real-time Methodology Evaluation

1. General Aspects

A. Identification

SA/SD - Real-time Structured Analysis/Structured Design by Yourdon, Inc.

Yourdon, Inc.
1133 Avenue of the Americas
New York, New York 10036
(212) 391-2828

B. Overview

SA/SD actually refers to two distinct methodologies: real-time and classic. This description covers the real-time version. The classic version is very similar, but it does not address real-time issues such as concurrency/synchronization or mapping to a distributed hardware architecture.

The methodology blends the three major requirements analysis techniques of state, flow, and object modeling with the design techniques of decomposition. Not only does SA/SD provide description of the techniques to use for requirements and design specification, but it also gives rules of thumb for applying the techniques in a reasonable and consistent manner both within and across life cycle phases.

The specific steps in software development as recommended by SA/SD are to construct a model:

i. of the context or environment of the system,

ii. of the internal behavior of the system,

iii. that shows the processor utilization of the system,

iv. that shows the software architecture utilization, and

v. that shows the coding architecture utilization.

Each model has both a physical and logical organization. The first two steps are requirements specification and analysis activities; the remaining three are design activities. Each model is differentiated by the type of behavior which

it describes and its constraining effect on the final implementation. As one proceeds through the steps, the final implementation becomes more constrained.

C. Life cycle phases supported:

All three (requirements, architectural and detailed design) phases are supported.

D. Software Categories

REALTIME

#	category
4	Procedure Control
5	Navigation
8	Message Processing
11	Simulation
15	Decision/planning aids
16	Pattern/image processing

CLASSIC

#	category
2	Event Control
3	Process Control
8	Message Processing
15	Decision and Planning Aids
16	Pattern and Image Processing

E. Suitable for systems of size: Any size.

2. Technical Aspects

A. Primary approach

Requirements specification is primarily done with a flow-oriented approach, but state transition diagrams also done. The design approach is decomposition.

B. Supports

	Capability
X	Traceability
X	Functional hierarchy/decomposition
X	Data hierarchy/data abstraction
X	Interface definition
X	Database definition
X	Data flow
X	Sequential control flow
X	Concurrency/parallelism
	Formal program verification
X	Iterative development (data dictionary)

C. Workproducts

They are indirectly relevant to MIL-STD documentation.

a. Textual

Structured specifications, data dictionary, mini-specs, state transition model, design specification for each code module, database design, operational *constraints*, physical constraints.

b. Graphical

Data flow, structure charts for code organization, data structures, finite state diagrams, decision tables, and control flow diagrams where state not independent.

D. Performance Specification

A timing constraint can be associated with a diagram.

E. Operating Qualities Specification

Man/machine interaction is partially addressed through data flow and state transition diagrams and prototype screens.

F. Ada compatibility

Ada Feature	Supported
Packages	X
Tasks	X
Generics	
Exception Handling	X
Types	
Representations	

Incompatibility with database modeling/design and with I/O.

G. Quality Assurance

SA/SD provides a set of rules and procedures to follow to check for coherence, correctness, clarity, and comprehensibility of specifications.

SA/SD recommends guidelines for manual validation via author/user reviews.

H. Independent of:

implementation language, hardware architecture, and operating system architecture. However, code organization, processor and software environment diagrams are drawn with a particular implementation environment in mind.

3. Support Aspects

A. Automated Tools

Three tool sets provide automated support for the SA/SD methodology: ARGUS, EXCELERATOR, and PROMOD. See their individual evaluations for more details.

B. Language

Uses both graphical and textual languages for all specification phases. The textual language is an informal structured English (a PDL in the case of detailed design); the graphical languages have formal syntax (symbols are provided) with informal semantics.

4. Management Aspects

SA/SD supports technical management through procedures for validation of workproducts and by providing guidelines for application of its techniques.

5. Usage Aspects

A. Equipment/Facilities Needed for SA/SD use:

See the specific tool set descriptions for ARGUS, EXCELERATOR, and PROMOD.

B. Usability

Level	Methodology
Easy to Use	
Moderately Easy to Use	X
Moderately Difficult to Use	
Difficult to Use	

C. Extent of Use

SA/SD is a mature technology which has been employed by many organizations for development of a wide variety of software projects.

6. Transferability

A. Availability: Commercially available.

B. Training Available

Yourdon, Inc offers public and private seminars and provides consultants. The public seminars cost approximately $900 per person and are scheduled nationally. Private seminars, which are structured as a five day course, can be arranged for up to 24 participants at a cost of roughly $8600.

C. Training and Experience Required

Training/Experience Needed			
months	USER	MANAGER	ORGANIZATION
< 1			
1 - 3	X	X	X
3 - 6			
> 6			

D. Primary Source of Documentation

Yourdon, Inc

4.7 SCR Methodology Evaluation

1. General Aspects

A. Identification

SCR - Software Cost Reduction Project

Naval Research Lab
Washington, DC 20375

B. Overview

The SCR requirements and design specification methodology is purely textual. It is based on the principles of information hiding and separation of concerns. Separation of concerns requires that information be divided into clearly distinct and relatively independent documents. Information hiding guides the architectural design of the software and leads to software that is easy to change.

The basic approach is data abstraction. Data items and the functions needed to create, store, retrieve, or manipulate them are identified. Event lists are used to document how the abstractions change relative to changing conditions as the software executes. This conditions can be nothing more complicated than passage of time.

The methodology includes procedural guidelines and suggested documentation formats that help keep the specifications complete and consistent.

C. Life cycle phases supported:

All specification phases supported.

D. Software Categories

#	category
2	Event Control
3	Process Control
4	Procedure Control
5	Navigation
6	Flight Dynamics
7	Orbital Dynamics
8	Message Processing
10	Sensor/signal Processing
11	Simulation
13	Data acquisition
15	Decision/planning aids
16	Pattern/image processing

E. Suitable for systems of size:

Any, but large systems would benefit from automated documentation control tools.

2. Technical Aspects

A. Primary approach

State-oriented for requirements since events and conditions are specified. Design approach is encapsulation by data abstraction.

B. Supports

	Capability
	Traceability
X	Functional hierarchy/decomposition
X	Data hierarchy/data abstraction
X	Interface definition
	Database definition
	Data flow
	Sequential control flow
X	Concurrency/parallelism
	Formal program verification
X	Iterative development

C. Workproducts

Satisfies the intent though not always the form of MIL-STD documentation.

a. Textual

Requirements document, module decomposition document, hierarchy subset (uses relationships between modules), process structure document, resource allocation document, and module interfaces.

b. Graphical

Has suggested formats for data item descriptions, and templates for value descriptions.

D. Performance Specification

Although the method is primarily textual, specification of timing constraints is expected. Accuracy is addressed in the formats for data item and value descriptions.

E. Operating Qualities Specification

Possible, since specifications are pure text.

F. Ada compatibility

Ada Feature	Supported
Packages	X
Tasks	X
Generics	X
Exception Handling	X
Types	X
Representations	X

G. Quality Assurance

The methodology suggests the use of a data dictionary for data item descriptions and the use of text macro expansion to keep definitions of functions and data items consistent across documents. Condition and event tables can be manually analyzed to check for consistency and completeness. Validation can be accomplished by manual review of the documents.

H. Independent of:

Hardware architecture, operating system architecture, and implementation language. An implementation language which enforces data abstraction is preferable (Ada does to some extent).

3. **Support Aspects**

A. Automated Tools

None specific to SCR, but document preparation and control tools are useful.

B. Language

Rigorous English is used for all specification phases.

C. Management Aspects

Addresses project management issues via manual procedures for document control. Technical (quality) management is addressed as analysis of event and condition tables.

4. Usage Aspects

A. Equipment/Facilities Needed for SCR use:

Text editor on any CPU.

B. Usability

Level	Methodology
Easy to Use	X
Moderately Easy to Use	
Moderately Difficult to Use	
Difficult to Use	

C. Extent of Use

The methodology has been used outside the Naval Research Lab by several organizations (see reference by Hester below) on several projects.

5. Transferability

A. Availability

In public domain in the form of technical reports from the Naval Research Lab.

B. Training Available

All documentation is in the public domain.

C. Training and Experience Required

Training/Experience Needed			
months	USER	MANAGER	ORGANIZATION
< 1	X	X	X
1 - 3			
3 - 6			
> 6			

D. Primary Source of Documentation

Naval Research Lab
Washington, DC 20375

Also see:

References

[Britton1981].
Kathryn Heninger Britton, R. Alan Parker, and David L. Parnas, "A Procedure for Designing Abstract Interfaces for Device Interface Modules," *Proceedings of 5th International Conference on Software Engineering*, pp. 195-204, March 1981.

[Chmura1982].
Louis J. Chmura and David M. Weiss, "The A-7E Software Requirements Document: Three Years of Change Data," *Proceedings from AGARD Conference CP-330*, September 1982.

[Heninger80].
Kathryn L. Heninger, "Specifying Software Requirements for Complex Systems: New Techniques and Their Application," *IEEE Transactions on Software Engineering*, vol. SE-6, no. 1, pp. 2-13, January 1980.

[Hester1981].
S.D. Hester, D.L. Parnas, and D.F. Utter, "Using Documentation as a Software Design Medium," *The Bell System Technical Journal*, pp. 1941-1977, October 1981.

[Parnas1972].
David L. Parnas, "On the Criteria to be Used in Decomposing Systems into Modules," *Communications of the ACM*, pp. 1053-1058, December 1972.

4.8 SREM Methodology Evaluation

1. General Aspects

A. Identification

SREM - Software Requirements Engineering Methodology

J. Mack Alford
TRW, Huntsville Laboratory
213 Wynn Drive
Huntsville, AL 35808
(205) 837-2400

B. Overview

SREM is based on a graph model of software requirements. The basic concept is that design-free functional software requirements should specify the required processing in terms of all possible responses (and the conditions for each response) to each input message across each interface. A message may contain input data or represent stimuli generated from an external event.

That is, the methodology is based on a stimulus/response approach as opposed to a purely hierarchical decomposition approach. The required actions of the software are expressible in terms of R_NETS (requirements networks) of processing steps. Each processing step is defined in terms of input data, output data, and the associated data transformation. The input interfaces (the system stimuli) are defined and the R_NETS trace the inputs through the various functional transformations to their associated system outputs (responses).

C. Life cycle phases supported:

SREM directly supports the requirements analysis phase; the other two phases are supported by DCDS. DCDS is the extension of SREM to design of distributed systems.

D. Software Categories

#	category
6	Flight Dynamics
7	Orbital Dynamics
10	Sensor/signal Processing
11	Simulation
13	Data acquisition

E. Suitable for systems of size:

Medium to large size.

2. Technical Aspects

A. Primary approach

State-oriented [alternatively referred to as finite state machine or stimulus-response descriptions].

B. Supports

	Capability
X	Traceability
X	Functional hierarchy/decomposition
X	Data hierarchy/data abstraction
X	Interface definition
X	Database definition
X	Data flow
X	Sequential control flow
X	Concurrency/parallelism
X	Formal program verification
X	Iterative development

C. Workproducts

They are indirectly relevant to MIL-STD documentation. The requirements and quality assurance sections of a specification are addressed by the RSL listing and the completeness and consistency tests REVS provides.

a. Textual

Documentation from queries to requirements database maintained by tool set REVS. Requirements are maintained as RSL statement listings.

b. Graphical

R-NET diagrams produced by tool set REVS, requires a Versetek printer.

D. Performance Specification

Both accuracy and timing constraints can be formally specified.

E. Operating Qualities Specification

Can be specified as UNSTRUCTURED_REQUIREMENTS. In some cases, RSL has been extended to cover specifying some of these qualities.

F. Ada compatibility

Ada Feature	Supported
Packages	X
Tasks	X
Generics	X
Exception Handling	X
Types	X
Representations	

G. Quality Assurance

Static analyses for consistency and completeness can be performed with the tool set REVS. Validation can be performed as a manual procedure of peer review, or as a dynamic simulation through REVS.

H. Independent of

implementation language, hardware architecture, and operating system architecture.

3. **Support Aspects**

A. Automated Tools

The set of automated tools is known as REVS. The tools aid in preparation of documentation of requirements, perform static consistency and completeness analyses of data flow and the database maintained by REVS, and perform dynamic analyses.

B. Language

Requirements Specification is done with RSL. RSL (Requirements Specification Language) has formal syntax with informal semantics. RSL is also extensible to allow the user to extend the language to meet the requirements of specific projects.

Architectural and detailed design can be done with the languages DCDS provides.

4. Management Aspects

SREM supports project management since it is possible to derive schedule information and management control information from the centralized data base (ASSM for Abstract System Semantic Model) REVS provides.

Technical management is supported through quality assurance tools REVS provides.

5. Usage Aspects

A. Equipment/Facilities Needed for REVS use:

CDC 7600 or Cyber 74/174/175 or VAX 11/780/VMS
graphics consoles
Pascal/Fortran compilers
plotters

B. Usability

Level	Methodology
Easy to Use	
Moderately Easy to Use	
Moderately Difficult to Use	X
Difficult to Use	

C. Extent of Use

SREM is mature, having been used on many projects by many organizations.

6. Transferability

A. Availability

In public domain.

B. Training Available

A description of the methodology (VOL 1) and a user's guide (VOL 2) are available as public documentation.

TRW will supply consultants on SREM (for a fee, of course). They also offer seminars on SREM.

C. Training and Experience Required

Training/Experience Needed			
months	USER	MANAGER	ORGANIZATION
< 1	X	X	
1 - 3			X
3 - 6			
> 6			

D. Primary Source of Documentation

Ballistic Missile Defense Advanced Technical Center
Huntsville, AL

SREM User's Group

[Bell1976].
Thomas E. Bell and David C. Bixler, *A Flow-oriented Requirements Statement Language*, TRW, April 1976.

[Rzepka1982].
William E. Rzepka, *Using SREM to Specify Command and Control Software Requirements*, RADC-TR-82-319, Rome Air Development Center, Griffiss AirForceBase, NY, 1982.

[Rzepka1983].
William E. Rzepka, "RADC SREM Evaluation Program - A Status Report," *ACM Sigsoft Software Engineering Notes*, vol. 8, no. 1, pp. 20-22, January 1983.

[Stone1983].
A. Stone, D. Hartschuh, and B. Castor, *SREM Evaluation*, Rome Air Development Center, February 1984.

4.9 VDM Methodology Evaluation

1. General Aspects

A. Identification

VDM - Vienna Development Method

Developed at the IBM Vienna Labs by Cliff Jones et al.
Vienna, Austria

Supported by Dines Bjorner
Department of Computer Science
Bldgs 343-344
Technical University of Denmark
DK--800Lyngby, Denmark

B. Overview

Define representation abstractions (syntactic domains); then create a series of refinements with justification from one refinement to the next. Based on set theory. The parts of a VDM specification are:

- state and type definitions - define the data objects that are to be employed within the specification and their logical types (one of sets, lists, mappings, and records);
- invariants - predicates on the state which assert that specific relationships hold between the values of the data objects in that state; and,
- operations and associated functions - mathematical formalisms that define key activities on the data objects, functions can not change the state of an object, operations can.

C. Life cycle phases supported: All.

D. Software Categories

#	category
1	Arithmetic Based
8	Message Processing
9	Diagnostic S/W
12	Database Management
14	Data presentation
15	Decision and Planning Aids
16	Pattern and image processing
17	Computer System Software
18	Software Development Tools

E. Suitable for systems of size:

Medium and Large, but experience in the methodology could best be gained by developing small systems first.

2. Technical Aspects

A. Primary approach

State-oriented for requirements; encapsulation for design.

B. Supports

	Capability
X	Traceability (mapping of GMB to SL1)
X	Functional hierarchy/decomposition
X	Data hierarchy/data abstraction
X	Interface definition
X	Database definition
	Data flow
	Sequential control flow
*	Concurrency/parallelism - *doesn't prohibit*
X	Formal program verification
X	Iterative development

C. Workproducts

Does not produce MIL-STD documentation.

a. Textual

Produces a series of more detailed formal specifications written in META-IV.

b. Graphical None.

D. Performance Specification

Can specify accuracy constraints through representations for types.

E. Operating Qualities Specification Not addressed.

F. Ada compatibility

Ada Feature	Supported
Packages	X
Tasks	X
Generics	X
Exception Handling	C
Types	X
Representations	

G. Quality Assurance

Verification is a manual procedure.

H. Independent of

implementation language, hardware architecture, and operating system architecture.

3. **Support Aspects**

A. Automated Tools

VDM has no set of automated tools.

B. Language

The language for all three specification phases is designated META-IV. It is based on set theory and is nonprocedural, with a formal syntax and semantics.

4. Management Aspects

VDM support technical management through its techniques for validation of the design.

5. Usage Aspects

A. Equipment/Facilities Needed for VDM use:

VDM has no automated tools, thus none needed.

B. Usability

A developer needs some knowledge of logic and the formalisms of finite mathematics.

Level	Methodology
Easy to Use	
Moderately Easy to Use	
Moderately Difficult to Use	X
Difficult to Use	

C. Extent of Use

VDM is mature, and has been applied to large projects by various organizations. DDC (Dansk Datamatik Center) used VDM to develop a validated Ada compiler.

6. Transferability

A. Availability

In public domain.

B. Training Available

Through seminars and consultants:
Dines Bjorner and Dansk Datamatik Center
Cliff Jones of Manchester University

As of December 1984, Dines Bjorner charges $2000/wk plus expenses for a seminar on VDM.

C. Training and Experience Required

Training/Experience Needed			
months	*USER*	*MANAGER*	*ORGANIZATION*
< 1			
1 - 3			
3 - 6	*X*		
> 6			*X*

D. Primary Source of Documentation

Also see:

References

[Bjorner1978].
The Vienna Development Method: The Meta-Language, Lecture Notes in Computer Science, Springer-Verlag, 1978.

[Bjorner1981].
Dines Bjorner, *The VDM Principles of Software Specification & Program Design*, Lecture Notes in Computer Science, Formalization of Programming Concepts, pp. 45-74.

[Clemmensen1984].
Geert B. Clemmensen and Ole N. Oest, *Formal Specification and Development of an Ada Compiler - A VDM Case Study*, IEEE, 1984.

[Jones1980].
Cliff B. Jones, *Software Development: A Rigorous Approach*, Prentice/Hall International, 1980.

[Shaw1981].
R.C. Shaw, P.N. Hudson, and N.W. Davis, "Introduction of a Formal Technique into a Software Development Environment," *ACM Software Engineering Notes*, vol. 9, no. 2, pp. 54-79, April 1984.

4.10 DCDS Methodology Evaluation

1. General Aspects

A. Identification

DCDS - Software Requirements Engineering Methodology

J. Mack Alford
TRW, Huntsville Laboratory
213 Wynn Drive
Huntsville, AL 35808
(205) 837-2400

B. Overview

DCDS is the extension of SREM to the design of distributed data processing systems with traceability to requirements. DCDS can be considered a two-part methodology:

- Programming-in-the-large - a data processor architecture is selected and the required processing is mapped onto it. This part, called Distributed Design, consists of allocating processing to a processor, allocating data to a processor, defining scheduling, and task design.
- Programming-in-the-small - algorithms are selected or constructed. This part is known as module design.

C. Life cycle phases supported:

SREM supports requirements analysis; DCDS supports architectural and detailed design.

D. Software Categories

#	category
6	Flight Dynamics
7	Orbital Dynamics
10	Sensor/signal Processing
11	Simulation
13	Data acquisition

E. Suitable for systems of size: medium and large.

2. Technical Aspects

A. Primary approach

The design approach is basically encapsulation.

B. Supports

	Capability
X	Traceability
	Functional hierarchy/decomposition
X	Data hierarchy/data abstraction
X	Interface definition
X	Database definition
X	Data flow
	Sequential control flow
X	Concurrency/parallelism - *including synchronization*
	Formal program verification
X	Iterative development

C. Workproducts

The specifications are maintained in the various languages used in DCDS. In that form, they would not be usable as MIL-STD documentation. However, it is possible that the tools generate usable documentation such as structure charts or tests plans from these statements.

D. Performance Specification

The constraints specified with SREM are taken into consideration. Also, the dynamic analysis tools can test whether the design meets those constraints.

E. Operating Qualities Specification

Fault-tolerant behavior is addressed in the detailed design phase.

F. Ada compatibility

Ada Feature	Supported
Packages	X
Tasks	X
Generics	X
Exception Handling	X
Types	X
Representations	X

G. Quality Assurance

The automated tools check for consistency and completeness. In fact, one checks for completeness of the processor allocation. Validation is done dynamically and can include testing whether the design satisfies performance constraints.

H. Independent of

implementation language, hardware architecture, and operating system architecture.

3. **Support Aspects**

A. Automated Tools

The tool set REVS from SREM has been extended for use with MDL (Module Design Language).

B. Language

Architectural design is done with a language DDL (Distributed Design Language) whose syntax is similar to RSL (SREM's language). Detailed design is done with a language MDL (Module Design Language) whose syntax is again similar to RSL.

DCDS also provides a language called TSL (Test Specification Language), which links requirements, design, and tests. TSL is used to define test plans and verify that those plans exhibit completeness of coverage.

4. Management Aspects

a. Project management

Modules and tasks are grouped into administrative units called units of code. Thus, they can be tracked by the central database, ASSM.

b. Technical management

In particular by test plan development.

5. Usage Aspects

A. Equipment/Facilities Needed for REVS use:

Unknown, probably same as REVS for SREM.

B. Usability

DCDS is still in the research phase, so usability is yet to be determined.

6. Transferability

A. Availability

DCDS is still under development, release planned for 1985.

B. Training Required

Can not be determined at this time (prior to release).

C. Primary Source of Documentation

Ballistic Missile Defense Advanced Technical Center
Huntsville, AL

[Alford1979].
Mack Alford, *Requirements For Distributed Data Processing Design*, IEEE, 1979.

[Alford1984].
Mack Alford, "SREM At the Age of Eight: The Distributed Computing Design System," Draft, December 1984.

4.11 JSD Methodology Evaluation

1. General Aspects

A. Identification

JSD - Jackson System Development

Michael Jackson Systems Limited
17 Conduit Street
London, England W1R9TD
Tel: 44-1-499-6655

B. Overview

JSD divides the process of software development into three main phases:

1. Modeling - an explicit examination of the external world with which the system will be concerned resulting in a diagrammatic description that clearly isolates and defines those aspects of the external world that are of interest. The model serves as an aid to understanding the subject matter of the system and provides the core for the formal specification of the system's functions.

2. Function - concerns the outputs of the system, what they should be and how they should be generated, resulting in diagrammatic descriptions attached to the functions in the previously defined model. The completed specification consists of a system specification diagram which defines the system as a set of logical processes communicating by data transfer and a set of structure diagrams which define the internal logic of each process.

3. Implementation - the system specification is converted into a form suitable for running on the chosen hardware by applying standard JSD procedures (called TRANSFORMATIONS) to package and realize the logical processes previously defined.

C. Life cycle phases supported: All.

D. Software Categories

#	category
1	Arithmetic-based
8	Message processing
9	Diagnostic Software
12	Database Management
14	Data presentation
15	Decision and planning aids
16	Pattern and image processing
17	Computer System Software
18	Software Development tools

E. Suitable for systems of size: All.

2. Technical Aspects

A. Primary approach

Object-oriented for requirements with a process being an encapsulation of local state that can communicate with other processes; the design approach is encapsulation.

B. Supports

	Capability
X	Traceability - *since structure charts associated with model*
	Functional hierarchy/decomposition
X	Data hierarchy/data abstraction
X	Interface definition
X	Database definition
X	Data flow
X	Sequential control flow
X	Concurrency/parallelism - *since each process can be implemented on a separate processor*
	Formal program verification
X	Iterative development

C. Workproducts

Data flow diagrams can be used for MIL-STD documentation.

a. Textual

Requirements are documented as a system specification that includes a model of the system to be developed. Structure texts (in the form of attribute grammars) detail the logic to be used within a process.

b. Graphical

Entity and Action lists
Tree-structured entity diagrams
data flow diagrams
database diagrams
system specification diagram (the MODEL)
complete system specification diagram with structure diagrams that describe functions.

D. Performance Specification Not addressed.

E. Operating Qualities Specification Not addressed.

F. Ada compatibility

Ada Feature	Supported
Packages	X
Tasks	
Generics	C
Exception Handling	
Types	
Representations	

G. Quality Assurance

Manual validation via structured walkthroughs and author/reader cycles.

H. Independent of

Transformations do not assume a specific implementation language, although the methodology has been primarily used on Cobol development projects. Also independent of hardware and operating system architecture.

3. Support Aspects

A. Automated Tools None.

B. Language

The languages used for all three phases are graphical. The transition from the graphics of detailed design to actual code is straightforward, since the graphics symbols map easily to implementation language control structures such as do...while.

4. Management Aspects

Technical management addressed through manual validation of workproducts.

5. Usage Aspects

A. Usability

Level	Methodology
Easy to Use	
Moderately Easy to Use	X
Moderately Difficult to Use	
Difficult to Use	

B. Extent of Use

JSD is a mature methodology that has seen extensive use in England.

6. Transferability

A. Availability

Commercially available.

B. Training Available

There is both public and proprietary documentation. Seminars and consultants are available.

Advanced Software Methods, Inc.
17021 Sioux Lane
Gaithersburg, MD 20878
(301) 948-1989
will provide consultancy and seminars. The course is five days and costs

approximately $6500 plus instructor's expenses.

A course in JSD has been offered through
Rocky Mountain Institute of Software Engineering
Aspen, Colorado.

C. Training Required

Training/Experience Needed			
months	USER	MANAGER	ORGANIZATION
< 1	X		
1 - 3			
3 - 6			X
> 6			

D. Primary Source of Documentation

Michael Jackson Systems Limited

Also see:

References

[Jackson1983].
Michael Jackson, *System Development*, Prentice/Hall International, 1983.

4.12 PAISLey Methodology Evaluation

1. General Aspects

A. Identification

PAISLey - Process-oriented, Applicative, Interpretable Specification Language

Pamela Zave
AT&T Bell Laboratories
Murray Hill, NJ 07974

B. Overview

PAISLey was developed explicitly for requirements specification of embedded real-time systems and takes an "operational" approach to requirements specification. That is, PAISLey allows the specifier to construct an executable model of the software as it would function in its environment.

The primary unit of specification is the process - a simple, abstract representation of autonomous digital computation. Each process is specified by supplying a "state space" (set of all possible states) and a "successor function" on that state space which defines the successor state for each state. A process is cyclic and goes through an infinite sequence of states (a distinguished "halted" state can be defined) asynchronously with other processes.

Because requirements are executable (by simulation), it is possible to attach and test timing constraints to processes. The constraints can be defined as maximum, minimum, mean, or constant evaluation time for the process.

Since PAISLey is an applicative language, a process can only access information in the state of another process by an explicit request. These requests are formulated as exchange functions and allow every form of synchronization to be defined.

C. Life cycle phases supported:

PAISLey only supports requirements specification. The design phases could be done with a methodology which also uses the concept of abstract processes (JSD for one).

D. Software Categories

#	category
2	Event Control
3	Process Control
4	Procedure Control
5	Navigation
6	Flight Dynamics
7	Orbital Dynamics
10	Sensor and signal processing
11	Simulation
13	Data acquisition

E. Suitable for systems of size: Any.

2. Technical Aspects

A. Primary approach

Object-oriented for requirements with a process being an encapsulation of local state that can communicate with other processes; has no specific design approach.

B. Supports

	Capability
	Traceability
	Functional hierarchy/decomposition
	Data hierarchy/data abstraction
X	Interface definition
	Database definition
X	Data flow
X	Sequential control flow
X	Concurrency/parallelism - *since each process can be implemented on a separate processor*
	Formal program verification
X	Iterative development

C. Workproducts

The only workproduct is the model of the system in the PAISLey language. As such, the model is not relevant to MIL-STD documentation.

D. Performance Specification

Both timing and accuracy constraints can be specified and tested.

E. Operating Qualities Specification

Man/machine interaction can be prototyped. The fault-tolerance and security properties of the specification can be tested since the model is fully executable as a simulation.

F. Ada compatibility

Ada Feature	Supported
Packages	X
Tasks	X
Generics	
Exception Handling	X
Types	
Representations	X

G. Quality Assurance

Some consistency and completeness properties are checked by the PAISLey language interpreter. Validation proceeds as a series of executions (simulations).

H. Independent of:

Implementation Language, hardware architecture, and operating system architecture.

3. Support Aspects

A. Automated Tools

PAISLey incorporates a language checker and a simulation facility, which interprets the PAISLey code.

B. Language

The PAISLey language is based on a class of programming languages designated applicative. Execution of a PAISLey program proceeds as applications or evaluations of functions rather than a series of subroutine calls. That is, PAISLey is more like LISP than like FORTRAN. One advantage of applicative languages is the the ease of mapping programs to distributed implementations.

4. Management Aspects

PAISLey supports technical (quality) management of the requirements analysis phase of a project through its simulation facility which results in a computerized validation of requirements.

5. Usage Aspects

A. Automated Tools

The primary tool is the PAISLey interpreter, others include a cross-referencer, a type checker, and a consistency checker. PAISLey assumes the processor is running UNIX. It also requires a text editor and a file system.

B. Usability

PAISLey is still in the research and development, so its usability is not established.

C. Extent of Use

PAISLey is still evolving and has only been used under the close supervision of its creator, Pamela Zave.

6. Transferability

A. Availability

Available on a case by case basis from Pamela Zave.

B. Training/Experience Required

Not established yet.

C. Primary Source of Documentation

Pamela Zave

Also see:

References

[Zave1982].
Pamela Zave, "An Operational Approach to Requirements Specification for Embedded Systems," *IEEE Transactions on Software Engineering*, vol. SE-8, no. 3, May 1982.

[Zave1983].
Pamela Zave, "Operational Specification Languages," *Proceedings ACM '83*, October 1983.

[Zave1984].
Pamela Zave, "An Overview of the PAISLey Project-1984," *ACM Sigsoft Software Engineering Notes*, pp. 12-19, July 1984.

4.13 SARA Methodology Evaluation

1. General Aspects

A. Identification

SARA - System ARchitect's Apprentice

Department of Computer Science
University of California at Los Angeles
Los Angeles, CA 90024

B. Overview

SARA is a set of modeling and evaluation tools that support a requirements-driven design methodology for concurrent systems. SARA encourages partition of a design or analysis universe into a system and its environment, with explicit models of their behaviors. The system includes tools to model behaviors (GMB), to model structures(SL1), and to model the structure of code-modules (MID).

GMB produces a graphics-based model of behavior in three domains: flow of control, flow of data, and interpretation. The model described in GMB may be interactively simulated and the control flow information can be formally analyzed for inconsistency, completeness, liveness (freedom from deadlock), and termination.

SL1 describes hierarchically related structures. A designer can specify a nested space of identifiers which partition a design universe and encapsulate behavioral models. A module encapsulates part of a behavioral model; a socket encapsulates behavior related to the interface between a module and its environment; an interconnection connects modules at their sockets and represents potential flow of data or control.

GMB models are mapped to SL1 structures, providing the connection between the behavior of objects in the actual environment and objects that will be instantiated during execution of the software system.

C. Life cycle phases supported: All.

D. Software Categories

#	category
2	Event Control
3	Process Control
4	Procedure Control
5	Navigation
6	Flight Dynamics
7	Orbital Dynamics
10	Sensor/signal Processing
13	Data acquisition

E. Suitable for systems of size: Any.

2. Technical Aspects

A. Primary approach

Flow-oriented for requirements; decomposition for design.

B. Supports

	Capability
X	Traceability *(mapping of GMB to SL1)*
X	Functional hierarchy/decomposition
X	Data hierarchy/data abstraction
X	Interface definition
	Database definition
X	Data flow
X	Sequential control flow
X	Concurrency/parallelism
*	Formal program verification *(not correctness)*
X	Iterative development

C. Workproducts

Since the requirements document is produced manually, it can be MIL-STD. Ada specification parts can be used as the formal description of modules and their interfaces.

a. Textual

Requirements document, reports from analyses, QA requirements document, and evaluation transcripts.

b. Graphical

Design models:
structural (SL1)
behavioral (GMB) with control and data flow
module interface (MID)

D. Performance Specification

Timing constraints can be associated with a socket.

E. Operating Qualities Specification None.

F. Ada compatibility

Ada Feature	Supported
Packages	*X*
Tasks	*X*
Generics	*X*
Exception Handling	*X*
Types	*X*
Representations	

G. Quality Assurance

Consistency and completeness of behavior models and module interfaces can be statically checked. Validation can be performed through a simulation.

H. Independent of

Although SARA was specifically designed for specification of concurrent systems, the specifications it produces are independent of hardware architecture, operating system architecture, and implementation language.

3. Support Aspects

A. Automated Tools

SARA includes a set of automated tools. There are language processors for GMB, SL1, and PLIP (a preprocessor for PL/1). The GMB models can be analyzed in the domain of control flow to identify deadlock states and critical transitions that lead to deadlock.

B. Language

a. Requirements Specification

GMB can model behavior of both the software system and its environment.

b. Architectural Design

SL1 models the necessary structures to instantiate the desired behavior captured in the GMB models. The MID models map from the SL1 structures to code modules. Because the structure of code modules is specified independently of the 'idealized' structure of the problem, analysis of various architectures for the code modules is encouraged.

c. Detailed Design

Can be done with Ada specification parts, or with PLIP, a PL/1 preprocessor.

4. Management Aspects

SARA's support for technical management is a quality assurance requirements document.

5. Usage Aspects

A. Equipment/Facilities Needed for SARA use:

Available for use under Berkeley Unix on a VAX or on MIT's MULTICS on ARPANET.

B. Usability

Level	Methodology
Easy to Use	
Moderately Easy to Use	
Moderately Difficult to Use	X
Difficult to Use	

C. Extent of Use

SARA has only been used at UCLA on student projects.

6. Transferability

A. Availability

In public domain.

B. Training and Experience Required

Training/Experience Needed			
months	USER	MANAGER	ORGANIZATION
< 1			
1 - 3	X	X	
3 - 6			
> 6			X

C. Primary Source of Documentation

Department of Computer Science
UCLA

Also see:

References

[Razouk1980].
Rami R. Razouk and Gerald Estrin, "Modeling and Verification of Communication Protocols in SARA: the X.21 Interface," *IEEE Transactions on Computers*, vol. C-29, no. 12, pp. 1038-1052, December 1980.

[Penedo81].
Maria Heloisa Penedo, Daniel M. Berry, and Gerald Estrin, "An Algorithm to Support Code-Skeleton Generation for Concurrent Systems," *Proceedings 5th International Conference on Software Engineering*, pp. 125-135, March 1981.

4.14 USE Methodology Evaluation

1. General Aspects

A. Identification

USE - User Software Engineering Methodology

Anthony I. Wasserman
Medical Information Science
University of California, San Francisco
San Francisco, CA 94143

B. Overview

USE is a methodology to support the development of specifications, designs, and implementations for interactive information systems. The automated tools are TDI (transition diagram interpreter), Troll (interface to a relational database system), RAPID (rapid prototypes of interactive dialogue), PLAIN (Programming LAnguage for INteraction), and the USE control system (management support),

The steps for requirements analysis are:

1. Identify system objectives and constraints, including conflicts of interest among user groups.

2. Model the existing system using a requirements analysis method (Structured Systems Analysis for instance).

3. Construct a conceptual model of the database, using the Semantic Hierarchy model of Smith and Smith.

4. Produce a system dictionary containing the names of all operations, all data items, and all data flows.

5. Review the analysis results within the development group and with the users and customers.

The formal requirements specification methodology is called BASIS (Behavioral Approach to the Specification of Information Systems). Each BASIS specification of a data abstraction includes three parts: the abstract image (representation), the invariant (behavioral characteristics that are always true), and input and output constraints (pre- and post-conditions) for

each operation defined on the abstraction. An informal narrative can be associated with each specification of an operation for an abstraction.

Architectural design constructs a structure chart of the overall design, following the general structure already outlined during the requirements phase. Detailed design is done via a program design language similar to Caine/Gordon. Both design phases are reviewed with walkthroughs.

C. *Life cycle phases supported:* All.

D. *Software Categories*

#	category
1	Arithmetic-based
9	Diagnostic Software
12	Database Management
14	Data presentation
17	Computer System Software
18	Software Development tools

E. *Suitable for systems of size:* Any.

2. Technical Aspects

A. *Primary approach*

Primarily object (both process and data) modeling for requirements analysis (data flow can be done), with functional decomposition (where the functions were previously identified in the requirements phase) for design.

B. Supports

	Capability
X	Traceability
X	Functional hierarchy/decomposition
X	Data hierarchy/data abstraction
X	Interface definition
X	Database definition
X	Data flow
X	Sequential control flow
*	Concurrency/parallelism *(doesn't prohibit)*
X	Formal program verification
X	Iterative development

C. Workproducts

The structure charts (of modularization) could be used for preparation of MIL-STD documentation.

a. Textual

Formal specifications of data abstractions and pdl definitions of modules.

b. Graphical

Entity-relationship diagrams, semantic hierarchy models, augmented transition diagrams for interactive dialogue, and structure charts for modularization.

D. Performance Specification None.

E. Operating Qualities Specification

Specification of the Man/machine interaction is directly addressed.

F. Ada compatibility

Ada Feature	Supported
Packages	X
Tasks	X
Generics	
Exception Handling	X
Types	
Representations	

G. Quality Assurance

Addressed by prototyping, consistency and completeness checking by automated tools, and by structured walkthroughs. The man-machine dialogue can be simulated.

H. Independent of:

hardware architecture, operating system architecture, and implementation language. Note however that some of the automated tools for code generation produce source code written in PLAIN.

3. Support Aspects

A. Automated Tools

The tools include code generation from PLAIN, consistency checkers, a database interface, and the control system for management of documentation and version control.

B. Language

a. Requirements Specification - nonprocedural specification language called BASIS

b. Architectural Design - informal structure charts

c. Detailed Design - uses a pseudocode pdl, with formal syntax and semiformal semantics.

4. Management Aspects

Project management is supported by the USE control system, which provides online information on the status of modules and the development process.

The USE control system also supports technical management by managing the documentation and can be used to enforce project standards. The other tools assist in system validation.

The USE control system supports configuration management by controlling versions of modules and systems.

5. Usage Aspects

A. Equipment/Facilities Needed for USE use:

The automated tools run under UNIX version 7 or BSD 4.1.

B. Usability

Level	Methodology
Easy to Use	
Moderately Easy to Use	X
Moderately Difficult to Use	
Difficult to Use	

C. Extent of Use

USE has only been used in a university setting.

6. Transferability

A. Availability

In public domain.

B. Training and Experience Required

Training/Experience Needed			
months	USER	MANAGER	ORGANIZATION
< 1			
1 - 3	X	X	
3 - 6			
> 6			X

C. Primary Source of Documentation

Anthony I Wasserman

Also see:

References

[Wasserman82].
Anthony I. Wasserman, "The User Software Engineering Methodology: an Overview," in *Information System Design Methodologies -- A Comparative Review*, ed. A.A. Verrign-Stuart, North Holland Publishing Company, 1982.

4.15 Tool Set Description Format

1. General Aspects

A. Identification

Gives the name and acronym of the tool or tool set and identifies the developing/supporting organization.

B. Methodologies

Lists what methodology the tool set supports.

C. Life cycle phases supported:

Identifies which of the specification phases the tool set supports:

— requirements analysis

— architectural design (intermodule communication, data structures)

— detailed design (module functionality)

D. Software Categories

Lists standard software categories which are compatible with this methodology.

#	Category	#	Category
1	Arithmetic-based	2	Event Control
3	Process Control	4	Procedure Control
5	Navigation	6	Flight Dynamics
7	Orbital Dynamics	8	Message Processing
9	Diagnostic S/W	10	Sensor/signal Processing
11	Simulation	12	Database Management
13	Data acquisition	14	Decision/planning aids
15	Data presentation	16	Pattern/image processing
17	Computer System Software	18	S/W development tools

E. Suitable for systems of size:

— Small (<2,000 lines of code) ?

— Medium (2,000 - 10,000 lines of code) ?

— Large (>10,000 lines of code) ?

2. Technical Aspects

A. Supports

Traceability
Functional hierarchy/decomposition
Data hierarchy/data abstraction
Interface definition
Database definition
Data flow
Sequential control flow
Concurrency/parallelism
Formal program verification
Iterative development

B. Workproducts

Are they relevant to MIL-STD documentation?

a. Textual

Description of reports, documents produced.

b. Graphical

Description of diagrams produced.

C. Performance Specification

Does the toolset have the capability to specify or test timing and/or accuracy constraints that apply to individual system functions?

D. Operating Qualities Specification

Does the toolset have the capability to specify the following constraints?

— Man/machine interaction

— Fault-tolerance

— Portability

— Reusability

— Security

E. Ada compatibility

Ada Feature	Supported
Packages	
Tasks	C
Generics	
Exception Handling	
Types	X
Representations	

X indicates support of feature.
C indicates conflict with feature.

F. Quality Assurance

How does the tool set support

a. Consistency checking ?

b. Completeness checking ?

c. Validation ? by a manual or computer-processed procedure ?

d. Rapid prototyping ?

Does it prototype the man-machine interface? the software modularization scheme? the functionality of the system? Is the execution mode of the prototype a simulation or a symbolic execution? Is the prototype suitable for pre-release?

e. Performance validation ? of correctness or efficiency?

3. Support Aspects

A. Degree of Integration

Vertical - within one phase of the software life cycle? Or horizontal - across more than one phase of the software life cycle?

B. Language

Identifies language(s) used for specification phases and its degree of formality.

— Requirements Specification

— Architectural Design

— Detailed Design

4. Management Aspects

Does the tool set support project, technical, or configuration management? How?

5. Usage Aspects

A. Equipment/Facilities Needed to use

Identify specific hardware and software (operating systems, graphics packages) required to use the tool set or associated automated tools.

B. Usability

Level	Methodology
Easy to Use	
Moderately Easy to Use	
Moderately Difficult to Use	
Difficult to Use	

C. Extent of Use

Has the tool set been used outside the developing organization? How much?

6. Transferability

A. Availability

Is the tool set in the public domain, commercially available, etc.?

B. Training Available

— Public documentation

— Proprietary documentation

— Consultants

— Seminars - scheduling and cost, if known

C. Training and Experience Required

Training/Experience Needed			
months	USER	MANAGER	ORGANIZATION
< 1			
1 - 3			
3 - 6			
> 6			

The table entries reflect the amount of training and experience time required to use the tool set effectively. A USER is an individual who develops or assists in developing requirements and/or design specifications. An ORGANIZATION is a group of users developing specifications as a team.

4.16 TAGS Tool Set Evaluation

1. General Aspects

A. Identification

TAGS

Teledyne Brown Engineering
Cummings Research Park
Huntsville, AL
(205)532-2036 -- Jerry Gotzold

B. Methodology

More elaborate version of SADT methodology.

C. Life cycle phases supported: All.

D. Software Categories

#	category
2	Event Control
3	Process Control
4	Procedure Control
5	Navigation
6	Flight Dynamics
7	Orbital Dynamics
8	Message Processing
10	Sensor and Signal Processing
11	Simulation
13	Data Acquisition

E. Suitable for systems of size: Any.

2. Technical Aspects

A. Supports

	Capability
	Traceability
X	Functional hierarchy/decomposition
X	Data hierarchy/data abstraction
X	Interface definition
X	*Database definition - as types*
X	Data flow
X	Sequential control flow
X	Concurrency/parallelism
	Formal program verification
X	Iterative development

B. Workproducts

They are not relevant to MIL-STD documentation.

a. Textual

The textual workproducts are tables that contain definitions for the input/output across an interface, for variables internal to a process block, and for constants internal to a process block. These tables have a standard format. All diagrams can be commented.

b. Graphical

Schematic block diagrams (SBD) which describe all components of the system and the data interfaces that connect them. Input/output relationships and timing diagrams (IORTD) which show the overall control flow for a single schematic block diagram component. Predefined process diagrams (PPD) which depict reiterated sequences of actions. Data structure diagrams (DSD) which graphically represent data defined in a parameter table.

C. Performance Specification

The ability to specify and test timing constraints is expected in third quarter 1985.

D. Operating Qualities Specification

Man/machine interaction is not directly addressed, but a human can be modeled as a process element.

E. Ada compatibility

Ada Feature	Supported
Packages	X
Tasks	X
Generics	X
Exception Handling	
Types	X
Representations	

F. Quality Assurance

There is a tool called a diagnostic analyzer that assists validation of the requirements. It does static analysis of individual diagrams for consistency and completeness. Consistency of interfaces and flows can also be checked.

G. Rapid prototyping

Prototyping of the functionality of the software system specified is expected in third quarter 1985. The tool will be generating Ada source code.

3. Support Aspects

A. Degree of Integration

The tools are well integrated both within and across phases of the software life cycle since they use a common database and the human interface is uniform across the tools.

B. Language

The diagrams use a formal language called IORL (Input/Output Requirements Language). IORL consists of mathematical expressions, types (character, numeric), and graphical symbols to represent processes and flows. Since the methodology is strictly top-down decomposition, IORL is used for all three phases.

4. Management Aspects

The tools directly support technical management (quality assurance) by static analysis, by maintaining a common project database, and by simulation of the design.

A tool for configuration management is expected to be released at the end of 1984.

5. Usage Aspects

A. *Equipment/Facilities Needed*

Tags requires APOLLO workstations and a VAX running VMS.

B. *Usability*

Level	Tools
Easy to Use	X
Moderately Easy to Use	
Moderately Difficult to Use	
Difficult to Use	

C. *Extent of Use*

TAGS is a new product, still undergoing development.

6. Transferability

A. *Availability*

TAGS is commercially available. A license for 1-8 workstations and 1 VAX is approximately $105,000.

B. *Training Available*

Teledyne Brown supplies proprietary documentation, consultants, and seminars (some of these items included in license fee).

C. *Training and Experience Required*

Training/Experience Needed			
months	USER	MANAGER	ORGANIZATION
< 1	X	X	
1 - 3			X
3 - 6			
> 6			

A developer needs a 2-3 week course which consists of an overview, details of using the tools, hands-on experience with the tools, and a walkthrough of a small system that was previously designed.

4.17 ARGUS II Tool set Evaluation

1. General Aspects

A. Identification

ARGUS

Boeing Computer Services Company
PO Box 24346
Seattle, WA 98124-0346
(206)

B. Methodology

A CAD/CAM software engineering environment for design that uses the Yourdon methodology.

C. Life cycle phases supported:

All specification phases are supported, however the requirements analysis phase is only supported by data flow diagrams.

D. Software Categories

Same as Classic Yourdon -

#	category
2	Event Control
3	Process Control
8	Message Processing
15	Decision and Planning Aids
16	Pattern and Image Processing

E. Suitable for systems of size: Any.

2. Technical Aspects

A. Supports

	Capability
	Traceability - *planned as future capability*
X	Functional hierarchy/decomposition
X	Data hierarchy/data abstraction
X	Interface definition
*	Database definition - *Via a data dictionary*
X	Data flow
X	Sequential control flow
*	Concurrency/parallelism -*planned as future support of Real-time Yourdon*
	Formal program verification
X	Iterative development

B. Workproducts

They are not directly usable as MIL-STD documentation.

a. Textual

Specification of modules in proscribed formats, customizable by project and PDL descriptions of module behavior.

b. Graphical

Data flow diagrams, structure charts.

C. Performance Specification

Not currently supported, planned as future capability.

D. Operating Qualities Specification

Man/machine interaction is directly addressed. Argus can produce sample report formats or screen layouts. The ability to specify security properties is planned as a future capability.

E. Ada compatibility

Although Ada is not directly supported, the resulting designs should have same compatibility with Ada as the Classic Yourdon methodology does.

F. Independent of

In theory ARGUS workproducts should be independent of the planned implementation language. However, the development tools for programmers support Fortran and Cobol.

3. Support Aspects

A. Degree of Integration

The tools are well integrated both within and across phases of the software life cycle since they use a common database and the human interface is uniform across the tools.

B. Language

The graphic notations of Yourdon methodology are used in requirements specification and architectural design. Detailed Design can be done with a pseudocode pdl.

4. Management Aspects

— Project Management

The management toolbox includes scheduling tools for controlling both projects and activities. A tool for tracing specific action items assists management in controlling activities across projects. An electronic spread sheet has proven extremely valuable to project managers in better controlling resource expenditures. Electronic scheduling and phone list capabilities are also provided. Various other managerial aids will be provided in future releases. {from "What About CAD/CAM for Software? The ARGUS Concept" by Leon G. Stucki, 1983 IEEE report # CH1919-0/83/0000/0129, pp 129-137}

— Technical management

The tools directly support quality assurance by static analysis, and by maintaining a common project database.

— Configuration management

Implemented, not yet released

5. Usage Aspects

A. Equipment/Facilities Needed

Currently, the use of ARGUS II requires an IBM PC-XT. ARGUS is, however, intended to be portable to other systems.

B. Usability

Level	Tools
Easy to Use	X
Moderately Easy to Use	
Moderately Difficult to Use	
Difficult to Use	

C. Extent of Use

ARGUS has been in continuous development since the late 1970's. It has only seen use within Boeing.

6. Transferability

A. Availability

Commercially available from Boeing Computer Services.

B. Primary Source of Documentation

Boeing Computer Services

C. Training and Experience Required

Training/Experience Needed			
months	USER	MANAGER	ORGANIZATION
< 1	X		
1 - 3			
3 - 6			
> 6			

4.18 EXCELERATOR Tool set Evaluation

1. General Aspects

A. Identification of Tool set

EXCELERATOR

Index Technology Corporation
Five Cambridge Center
Cambridge, MA 02142
(617) 491-7380

B. Methodology

Classic Yourdon.

C. Life cycle phases supported: All.

D. Software Categories

Same as Classic Yourdon -

#	category
2	Event Control
3	Process Control
8	Message Processing
15	Decision and Planning Aids
16	Pattern and Image Processing

E. Suitable for systems of size: Any.

2. Technical Aspects

A. Supports

	Capability
	Traceability
X	Functional hierarchy/decomposition
X	Data hierarchy/data abstraction
X	Interface definition
X	Database definition - *Via a data dictionary*
X	Data flow
X	Sequential control flow
	Concurrency/parallelism
	Formal program verification
X	Iterative development

B. Workproducts

The workproducts are not directly usable as MIL-STD documentation.

a. Textual

Can generate textual mini-specs (for one module) for project reviews, and report and screen mockups.

b. Graphical

Data flow diagrams, minispecs (logic within a module), data model diagrams, structure charts.

C. Operating Qualities Specification

Man/machine interaction is directly addressed with report and screen mockups.

D. Ada compatibility

Does not directly support Ada, but compatibility of designs should be the same as using the classic Yourdon methodology.

E. Quality Assurance

Consistency is only assured for items described by the data dictionary. One of the automated tools does completeness and consistency checking of data flow diagrams.

3. Support Aspects

A. Degree of Integration

The tools are well-integrated within a phase.

B. Language

Requirements Specification and Architectural Design are done with the graphic notation of Yourdon methodology.

4. Management Aspects

Technical management is supported by a common project data dictionary that assists manual procedures for quality assurance. Tools that address project management are planned for the future.

5. Usage Aspects

A. Equipment/Facilities Needed

Use of EXCELERATOR requires either an IBM PC-XT, 3270-PC or PC-AT.

B. Usability

Level	Tools
Easy to Use	X
Moderately Easy to Use	
Moderately Difficult to Use	
Difficult to Use	

C. Extent of Use

EXCELERATOR is a commercial product. It has been used outside of Index Technology Corp., its developer, on more than ten projects.

6. Transferability

A. Availability

EXCELERATOR is commercially available through Index Technology Corporation or local IBM branch offices.

B. Primary Source of Documentation

Index Technology Corporation

C. Training and Experience Required

Training/Experience Needed			
months	USER	MANAGER	ORGANIZATION
< 1	X		
1 - 3			
3 - 6			
> 6			

4.19 PROMOD Tool set Evaluation

1. General Aspects

A. Identification

PROMOD

GEI Systems House
Albert Einstein Strasse - 61
D-5100 Aschen, Germany
Tel: 02408/130

B. Methodologies

PROMOD uses classic Yourdon for requirements analysis and a combination of Yourdon Structured Design and the principle of information hiding for design. As a notation for detailed design, it is possible to use use a pseudo-code (like Caine-Farber-Gordon) pdl or or the Jackson Structured Programming notation.

C. Life cycle phases supported: All.

D. Software Categories

Same as Classic Yourdon -

#	category
2	Event Control
3	Process Control
8	Message Processing
15	Decision and Planning Aids
16	Pattern and Image Processing

E. Suitable for systems of size: Any.

2. Technical Aspects

A. Supports

	Capability
	Traceability
X	Functional hierarchy/decomposition
X	Data hierarchy/data abstraction
X	Interface definition
X	Database definition - *Via a data dictionary*
X	Data flow
X	Sequential control flow
X	Concurrency/parallelism - *can specify synchronization*
	Formal program verification
X	Iterative development

B. Workproducts

They are not in MIL-STD form, although some of the graphical output can be used to develop MIL-STD documentation.

The graphical output includes:
Data flow diagrams, minispecs (logic within a module), Nassi-Schneidermann charts if desired, call trees, and data hierarchy trees.

The only textual workproduct would be any pseudocode produced for detailed design.

C. Ada compatibility

PROMOD does not directly support Ada but resulting designs can be compatible in the following ways:

Ada Feature	Supported
Packages	X
Tasks	X
Generics	X
Exception Handling	
Types	X
Representations	

D. Quality Assurance

Consistency checking is supported by syntax directed editing along with syntax and semantic checking. There is an analysis for checking completeness.

3. Support Aspects

A. Degree of Integration

The tools are well integrated both within and across phases of the software life cycle since they use a common set of VMS files and the human interface is uniform across the tools.

B. Language

Requirements Specification is done with the graphic notation of Yourdon methodology. Architectural Design is done with the graphic notation of Yourdon methodology. Detailed Design is done with either a pseudocode pdl, or the JSP notation, or Nassi-Schneidermann charts.

4. Management Aspects

Project management is based on a project model and a project database. These can be used to manually produce a work breakdown structure and milestone definition. Tools to support these tasks are expected in the future.

Technical management is based on tools that directly support quality assurance by static analysis, and by maintaining a common project file structure. The methodology recommends metrics for quality assurance ala Yourdon. The transition from phase to phase is supported by the project file structure.

5. Usage Aspects

A. Equipment/Facilities Needed

PROMOD requires a VAX running VMS or an IBM or Victor CPU.

B. Usability

Level	Tools
Easy to Use	
Moderately Easy to Use	
Moderately Difficult to Use	X
Difficult to Use	

C. Extent of Use

PROMOD has been used outside the developing organization on more than 10 projects.

6. Transferability

A. Availability

Commercially available.

B. Training Available

There is both public and proprietary documentation available on PROMOD from its developing organization, GEI.

C. Training and Experience Required

Training/Experience Needed			
months	USER	MANAGER	ORGANIZATION
< 1	X		
1 - 3			
3 - 6			
> 6			

4.20 PSL/PSA Tool Evaluation

1. General Aspects

A. Identification

PSL/PSA - Problem Statement Language/Problem Statement Analyzer

Dan Teicherow
ISDOS Project
University of Michigan
Ann Arbor, MI 48109

B. Methodologies

Basically documents data flows, whether for requirements or design analysis. Can be used with any methodology that uses data flow diagrams.

C. Life cycle phases supported:

Supports data flow analysis of requirements analysis and architectural design phase. Can maintain some module documentation.

D. Software Categories

Depends on methodology PSL/PSA is used with.

E. Suitable for systems of size: Medium and Large.

2. Technical Aspects

A. Supports

	Capability
X	Traceability
	Functional hierarchy/decomposition
X	Data hierarchy
X	Interface definition
*	Database definition - *Via a data dictionary*
X	Data flow
*	Sequential control flow - *has an IS TRIGGERED BY attribute*
	Concurrency/parallelism
	Formal program verification
X	Iterative development

B. Workproducts

Extension of the basic tool can provide reports consistent with MIL-STD documentation [The Hughes' version known as ASAT has this capability].

a. Textual

The contents of the database PSL/PSA maintains can be reported in various ways.

b. Graphical

Data flow diagrams.

C. Ada compatibility Not applicable.

D. Quality Assurance

The tool maintains consistency of the information in the database and records when updates are effected. The database can be analyzed to find dangling entries or definitions not used anywhere. The PSA portion of the tool can be used to attempt to verify properties concerning the information and data flows maintained by the database.

E. Independent of

Implementation language, hardware architecture, and operating system architecture.

3. Support Aspects

A. Degree of Integration

There is basically only one tool.

B. Language

PSL (Problem Statement Language) is basically a data base manipulation language with predefined keywords to represent the relationships the data-base is maintaining. The syntax is formal; the semantics are semi-formal.

4. Management Aspects

The reports that can be generated from the database can be used for checking some of a project's progress against milestones.

The consistency of data flow can be checked by the tool which analyzes data flows.

The database can contain responsible person information.

5. Usage Aspects

A. Equipment/Facilities Needed

PSL/PSA is available on a wide range of CPUs. Note that its computational requirements are quite heavy.

B. Usability

Level	Tools
Easy to Use	
Moderately Easy to Use	X
Moderately Difficult to Use	
Difficult to Use	

C. Extent of Use

PSL/PSA is a mature software product. It has been used on a large number of information systems development projects by a large number of different organizations.

6. Transferability

A. Availability

PSL/PSA is commercially available for about $40,000 and there is a government owned version called CADSAT.

B. Training Available

Proprietary documentation, consultants, and training courses are available from the ISDOS Project at the University of Michigan in Ann Arbor.

C. Training and Experience Required

Training/Experience Needed			
months	USER	MANAGER	ORGANIZATION
< 1		X	
1 - 3	X		
3 - 6			X
> 6			

D. Primary Source of Documentation

Dan Teicherow

5.0 SOFTWARE ACQUISITION LIFE CYCLE

5.1 INTRODUCTION

This section provides brief paragraphs describing two specific standards for the software acquisition life cycle: AFR 800-14 and MIL-STD-SDS. Currently, AFR 800-14 is the standard for Air Force software acquisition. MIL-STD-SDS is in final stages of development as the DoD standard for software acquisition. The two standards do not conflict with one another, although they do use different names for their life cycle phases.

The intention of this guidebook is to aid the project manager in selecting a methodology and support tools for a particular system category and life cycle phase. Additionally, the guidebook will aid the project manager in deciding which methodologies will support the specification activities of the remaining life cycle phases. The guidelines will assist the software development manager in selecting specification technologies and tools to assist in meeting the specification needs of the system requirements, software requirements, and software design activities.

The following paragraphs describe the two life-cycle standards mentioned above and, further, describe how the requirements and design activities are supported by the methodologies in this guidebook.

5.2 AFR 800-14 SYSTEM ACQUISITION LIFE CYCLE

The system acquisition life cycle categorizes overall program management activities. Its five phases extend from program conception to termination. It can be thought of as the larger system framework within which the software development life cycle takes place. The five phases of system acquisition are shown in figure 5-1.

5.3 AFR 800-14 SOFTWARE DEVELOPMENT LIFE CYCLE

The software development life cycle fits within the larger program framework of the system acquisition life cycle and may occur within one, or span a number of system acquisition life-cycle phases. Specifically, software development comprises six phases: Analysis, Design, Coding and Checkout, Test and Integration, Installation, and Operation and Support. These six phases and their definitions are shown in figure 5-2.

AFR 800-14 SYSTEM ACQUISITION LIFE CYCLE				
PHASE 1	**PHASE 2**	**PHASE 3**	**PHASE 4**	**PHASE 5**
CONCEPTUAL	**VALIDATION**	**FULL-SCALE DEVELOPMENT**	**PRODUCTION**	**DEPLOYMENT**

Conceptual:	In which solutions to problems are planned, refined, and alternative solutions conceived. Preliminary requirements are formulated.
Validation:	In which major system characteristics are refined through studies, preliminary modeling, computer program development, etc., to validate the choice of alternatives and to decide whether to proceed into the next phase.
Full-Scale Development:	In which all the major items comprising the system(s) are designed, fabricated, tested, and integrated. The operation of the system closely resembles the operation of the production system.
Production:	In which all the production systems are completed, delivered, and accepted.
Deployment:	Lasts from the time the first system becomes operational until the last system is removed from operational inventory.

Figure 5-1. AFR 800-14 System Acquisition Life Cycle.

AFR 800-14 SOFTWARE DEVELOPMENT LIFE CYCLE					
PHASE 1	**PHASE 2**	**PHASE 3**	**PHASE 4**	**PHASE 5**	**PHASE 6**
REQUIREMENTS ANALYSIS	**DESIGN**	**CODING/ CHECK-OUT**	**TEST/ INTEGRATION**	**INSTALLATION**	**OPERATION/ SUPPORT**

Requirements Analysis:	Defines functional, interface, and performance requirements for software.
Design:	Develops a design approach, including mathematical models, functional flow charts, and detail flow charts. Also defines relationships among components.
Coding and Checkout:	Coding translates flow charts into computer programs and data. Checkout converts intial code and data into an operational computer program. The software is operational when it uses predefined inputs and produces correct outputs.
Test and Integration:	Tests the computer program against requirements in the development specification. Tests all of the computer program, including individual computer program function or module tests through total computer program formal qualification tests.
Installation:	Includes loading and running of computer programs after successful qualification and integration.
Operation and Support:	Assesses the operational suitability of the system.

Figure 5-2. AFR 800-14 Software Development Life Cycle.

5.4 DOD-STD-SDS COMPUTER SOFTWARE DEVELOPMENT LIFE CYCLE

The DOD-STD-SDS (a draft SDS Documentation Set has been released that contains DOD-STD-2167, MIL-STD-483, MIL-STD-490, and MIL-STD-1521) is a standard that establishes requirements to be applied during the development and acquisition of Mission-Critical Computer System (MCCS) software, as defined in DOD Directive 5000.29. The standard may also be applied to non-MCCS software development and acquisition.

The DOD-STD-SDS standard comprises six phases: (1) Software Requirements Analysis, (2) Preliminary Design, (3) Detailed Design, (4) Coding and Unit Testing, (5) CSC Integration and Testing, and (6) CSCI-Level Testing. All analysis performed prior to phase 1 is termed Pre-Software Development. The six phases and their relationship to the basic reviews that occur during the software development life cycle are shown in figure 5-3.

5.5 RELATION OF METHODOLOGIES TO LIFE CYCLE PHASES

The guidelines in this document are intended as an aid to the project manager during the requirements and design phases of the software acquisition life cycle. Requirements analysis can be broken down into two components: system requirements analysis and software requirements analysis. Software design is a single phase.

Figure 5-4 shows the relationship of the methodologies described in section 4.0 of this document and the requirements and design life-cycle phases.

DOD-STD-SDS SOFTWARE DEVELOPMENT LIFE CYCLE					
PHASE 1	**PHASE 2**	**PHASE 3**	**PHASE 4**	**PHASE 5**	**PHASE 6**
SOFTWARE REQUIRE-MENTS ANALYSIS	**PRELIM-INARY DESIGN**	**DETAILED DESIGN**	**CODING AND UNIT TESTING**	**CSC INTE GRATION/ TESTING**	**CSCI TESTING**

Software Requirements Analysis:	Define and analyze functional, performance, interface, and qualification requirements for each CSCI.
Preliminary Design:	Develop a top-level design of each CSCI which completely reflects the requirements specified in the SRS and IRS(s).
Detailed Design:	Develop a modular, detailed design for each CSCI.
Coding and Unit Testing:	Code and test each unit making up the detailed design.
CSC Integration and Testing:	Integrate units of code entered in the developmental configuration and perform informal tests on aggregates of integrated units.
CSCI Testing:	Conduct formal tests on each CSCI to show that the CSCI satisfies its specified requirements. Record and analyze test results.

Figure 5-3: DoD-STD-SDS Software Development Life Cycle

		LIFE CYCLE PHASE		
		REQUIREMENTS	**DESIGN**	**METHODOLOGY**
METHODOLOGY KEY	A	X	X	DSSD
	B		X	HDM
	C	X	*	SADT
	D	X	X	SA/SD
	E	X	X	SCR
	F	X		SREM
	G	X	X	VDM
	H		X	DCDS
	I	X	X	JSD
	J	X		PAISLey
	K	X	X	SARA
	L	X	X	USE

where X = yes, covers this phase.
* = partial coverage.

Figure 5-4 Life Cycle Phase Coverage

6.0 SAMPLE PARAGRAPHS FOR STATEMENTS OF WORK

6.1 INTRODUCTION

This section provides sample statement of work paragraphs, which are examples of the various approaches that can be used to specify the use of software requirements methodologies or software design methodologies with the aid of the Specification Technology Guidebook. The sample SOW paragraphs are written to allow various degrees of constraint in the imposition of requirements and design methodologies, from tight constraint to mild guidance. They may need to be modified or specific details added by the Air Force acquisition manager to fit the development environment of a particular project or the contractual relationship being considered.

The information in parentheses (...) must be filled in by the acquisition manager. The SOW paragraphs require that "implementation details" be provided by the Computer Program Development Plan; but the acquisition manager may choose, in his specific contractual arrangement, to use another CDRL item in which to request this detail. Existing data item descriptions (DID) for the specification and development of software may not include provisions for the types of information generated by the selected/specified methodology. Therefore, the DID may need to be modified to provide a section for the newly required information.

For tightly constrained requirements (sections 6.2 and 6.3), reference the desired methodologies in section 4.0 by name and section number.

6.2 TIGHTLY CONSTRAINED--DIRECT SPECIFICATION

The following paragraph can be used to specify specific methodologies or techniques identified in the guidebook. The guidebook would thus serve AF personnel in selecting specification methodologies for mission software products. The methodology for each software product, such as ground support software and in-flight software, would be determined separately.

Tightly Constrained--Direct Specification

The computer program (...requirements/design...) shall be specified in accordance with techniques described in the Specification Technology Guidebook, RADC-TR-85-XX. The following methodologies shall be used:

Methodology	*Guidebook Section No.*	*Phase(s)*

The methodologies and techniques, and necessary details, shall be described in the Computer Program Development plan (see CDRL).

6.3 TIGHTLY CONSTRAINED--SUBSET SPECIFICATION

The following paragraph can be used to specify a minimum set of techniques plus the use of the guidebook by a contractor to select additional specification or design techniques for the entire software project or an individual computer program.

Tightly Constrained - Subset Specification

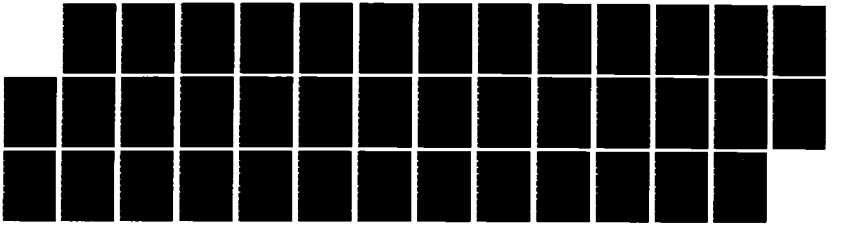
The computer program (...requirements/design...) shall be (...specified/designed...) with techniques described in the Specification Technology Guidebook RADC-TR-85-XX. As a minimum, the following specification (or design) methodologies shall be used:

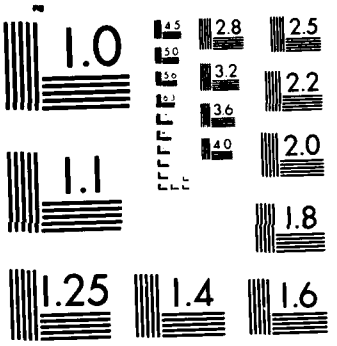
Methodology	*Guidebook Section No.*	*Phase(s)*

Selection of additional techniques or methodologies, using an overall significance level (OSL) of (...) shall be performed in accordance with procedures described in section 2.0 of the Specifications Technology Guidebook. The resulting set of methodologies and necessary implementation details shall be described in the Computer Program Development Plan (see CDRL). Rationale for the selection of those techniques, from the candidate set identified by the Specification Guidebook, shall be provided.

AD-A162 457 SPECIFICATION TECHNOLOGY GUIDEBOOK(U) BOEING AEROSPACE 3/3
CO SEATTLE WA D R ADDLEMAN ET AL AUG 85
RADC-TR-85-135 F30602-84-C-0073

UNCLASSIFIED F/G 9/2 NL





6.4 MODERATELY CONSTRAINED SPECIFICATION

The following paragraph can be used to specify the use of the guidebook for selecting software requirements and design techniques. An overall significance level (OSL) is the only predetermined factor. The paragraph does not constrain the developer to use specific methodologies, nor does it identify specific phases during which the methodologies are to be applied.

Moderately Constrained

> *The computer program (...requirements/designs...) shall be (...specified/designed...) in accordance with the methodologies described in the Specification Technology Guidebook, RADC-TR-85-XX. Selection of the methodologies, using an overall significance level (OSL)of (...) shall be performed in accordance with procedures described in section 2.0 of the Specification Technology Guidebook. The resulting methodology, and necessary implementation details, shall be described in the Computer Program Development Plan (see CDRL). Rationale for the selection of the methodology, from the candidate set identified by the Specification Technology Guidebook, shall be provided.*

6.5 LOOSELY CONSTRAINED SPECIFICATION

The following paragraph can be used to allow the contractor extensive freedom in selecting software requirements and design methodologies.

Loosely Constrained

> *The computer program (...requirements, designs...) shall be developed using methodologies described in the Specification Technology Guidebook, RADC-TR-85-XX. Selection of techniques shall be in accordance with procedures described in section 2.0 of the Specification Technology Guidebook. The resulting set of methodologies and techniques, shall be described in the Computer Program Development Plan (see CDRL). Rationale for the selection of those methodologies and techniques, from the candidate set identified by the Specification Technology Guidebook, shall be provided.*

APPENDIX A: ARMAMENT

The software usage described in this appendix is based on a representative site devoted largely to this mission. Therefore, the procedures covered in this appendix many not include all aspects of the armament mission.

A1. ARMAMENT DIVISION

The Armament Division (AD) is a developing agency for tactical weapon systems, and particularly for threat, missile, and scoring systems. All embedded software systems development at AD is performed by contractors. The contractors are usually small, specialized, high-technology companies, but larger aerospace companies also contribute to the systems development. The software contained in these systems typically tends not to be critical, even though the systems themselves may be critical. The contractors design, develop, and test the software according to contractor-defined standards and under the general contractual-level supervision of Air Force personnel. Testing practices vary widely among the many contractors supporting the AD.

A2. AD MISSION

The primary mission area applicable to the Armament Division is armament. Secondary mission areas are aeronautical and missile/space, including avionics systems and air-to-air and air-to-ground missile systems.

A3. SOFTWARE DEVELOPMENT ENVIRONMENT

The most significant category of software is the operational software for embedded systems. The embedded systems developed at AD are generally found in three categories:

a. Threat systems. Defensive and offensive radar, targeting and tracking systems, and electronic countermeasures.

b. Missile systems. Air-to-air and air-to-ground missile systems.

c. Scoring systems. Proximity detectors for projectiles, used in aerial gunnery training.

The data processing and administrative software areas was not surveyed; typically, these systems are in place and are not subject to extensive new development. Also, the embedded operational software systems are more germane and critical to the primary mission of the AD.

General Environment. Software development is conducted as a part of embedded systems development. System requirements are defined by Air Force personnel and then the systems and the software are designed, implemented, and tested by contractors selected in competitive bidding. The Air Force may participate in the definition and specification of detailed requirements. The companies range from small businesses to major aerospace corporations. Procurement requirements for software are similar for all systems development and their implementation is monitored by design reviews, audits, and detailed reviews of contractor-furnished documentation.

Software (computer programs and associated data) typically comprises a significant share of the contractors' system development effort (labor), ranging up to one-half or even more of the total effort.

Standards. The Armament Division complies with the 800-series Air Force regulation for systems development. The following regulations and standards are applicable to embedded systems and software development:

- AFR 80-14 (Test and Evaluation)
- AFR-800 (Management of Computer Resources in Systems)
- MIL-STD-483 (Configuration Management Practices)
- MIL-STD-1521A (Reviews and Audits)
- MIL-STD-490 (Specification Practices)
- MIL-STD-1750A (16-Bit Instruction Set Architecture)
- MIL-STD-1589A (JOVIAL J73 Language)
- MIL-STD-1815 (Ada Language)
- IEEE STD-716 (ATLAS Language)
- American National Standard X3.9 (FORTRAN 77 Language)

Administration. Contractors are required to identify and account for embedded software as computer program configuration items (CPCI), including support computer programs (such as ATE software). The programs are controlled using allocated configuration identification and product configuration identification, with associated Part I and Part II specifications. CPCI-to CPCI and CPCI-to-hardware interface specifications are also required. Computer programs are subjected to a sequence of reviews and audits: preliminary design review, critical design review, functional configuration audit, and physical configuration audit.

Languages. Approved high-order languages are mandated for embedded computer programs. The specified languages are JOVIAL J73, Ada, and FORTRAN 77, in that order of priority. The specified language for automatic test equipment (ATE) is ATLAS. Exceptions to the priority of the approved list must be authorized; the use of assembly language or nonapproved higher order language (HOL) subsegments must also be authorized.

Support Software. Contractors are encouraged to use off-the-shelf components and support tools. The following support tools are required:

a. An efficient compiler (in terms of code generated) and code generator for an approved HOL.

b. A software development station with aids, including a programmable read only memory (PROM) programmer, if applicable.

c. A complete support software library, including but not limited to an editor, linking loader, and run-time support routines.

d. Compatible hardware and software peripheral equipment

Capacity Requirements. Embedded software is required to have a 30% spare capacity in memory utilization. Also, the software is required to exercise only 70% of the computer's throughput and input/output channel capacity.

Coding Standards. Contractors must establish coding standards for software development. The following minimum requirements are imposed:

a. Modularity. Computer programs shall be modular in design. Module identification shall be along functional lines with ease of maintenance being a prime consideration. To the maximum extent practical, data base information shall not be provided as in-line code. Rather, data shall be provided in a separate, non-executable module or file.

b. Structured Programming. The principles of top-down, structured programming shall be used to the maximum extent practical. Each module or submodule of the computer program shall be designed with a single entry point and a single exit point.

c. Comments. Computer program listings shall contain comments that completely describe the functions being performed in each program module.

A4. SOFTWARE CHARACTERISTICS

The most common categories of software developed are embedded operational programs and ground support systems, particularly for system tests using ATE. The computer programs range from small (under 16K statements) to large (64K to 200K statements), the project size ranges from small to medium, and the development periods are relatively short (between 1 and 3 years). Emphasis is placed on the adequacy of documentation, with the following document items being representative:

- System specification
- Computer program development specification (Part I)
- Computer program product specification (Part II)
- Computer program development plan (CPCD)
- Configuration management plan
- Interface control document
- Operator's manual
- User's manual
- Computer program test plan and procedures

The criticality of the computer programs developed at this site ranges from zero to two. Ground support and system test programs are considered criticality zero, while operational programs are either criticality one or two. Missile and weapon systems software and flight control systems software are considered more critical than telemetry, simulation, display, and scoring calculation programs. However, no distinction was evident in the level of requirements for criticality one or two software.

Only general information on the characteristics of the software was obtained for the three categories of systems developed at the site (threat, missile, and scoring systems). These are discussed in the following paragraphs.

A1.1 Threat Systems

The principal languages used in program implementation are FORTRAN-77 and some assembly language routines for special processes, such as input/output handling. The software for these systems has been typically programmed on minicomputers, such as the ECLIPSE, NOVA-3, VAX-11, HP 1000, ROLM, and PDP-

11/LSI-11. The system functions they perform include ground systems, antenna control, network interfacing for mission data, servo/slave control, and target scenario simulation (used in electronic warfare air crew training). Representative functions that are implemented in software include the following:

- Interface to keyboard, CRT, and disk.
- Radar ranging and position calculation.
- Servo positioning.
- Message handling.
- Radar systems monitoring.
- Console control interface.
- Radar systems simulation.
- Tracking control.
- Target and threat data interpretation.

A4.2 Missile Systems

The principal languages used for embedded operational computer programs are JOVIAL, J73, and assembler. Because of the throughput performance requirements and limitations on available onboard memory inherent to missile systems, assembly language is commonly used to program the missile-resident computer programs. JOVIAL is used in non-missile-resident software and ATLAS is used for ground checkout programs. The onboard programs typically occupy 40K to 62K bytes of memory. The functions performed by the onboard programs include the following:

- Navigation
- Autopilot
- Executive control
- Guidance

- Tracking and stabilization
- Fusing
- Downlink telemetry
- Electronic counter-countermeasures
- Built-in-test

The ground systems developed for missile contracts performs the following functions:

- Data link interface
- Command receiver
- Radar processing
- Ground test

The ground test (ATE) software is developed and executed on minicomputers, such as the PDP-11; flight software is targeted for execution on a special-purpose 16-bit microprocessor, such as the General Missile Processor.

A4.3 Scoring Systems

Examples of scoring systems are the Digital Doppler Scoring System, Antenna Identification Scoring system, and Aerial Gunnery Target System. These systems typically are targeted for 8-bit or 16-bit microprocessors, such as the Intel 8080 and 8086. They are often coded in assembly language, using the Intel Development System. The software functions performed encompass the following:

- Graphics
- Trajectory calculation
- Performance and statistics calculation

- Trajectory calculation algorithms
- Analog-to-digital conversion
- Telemetry (discrete and continuous)
- Front end formatting and filtering

A5. CATEGORIZATION OF SOFTWARE FUNCTIONS

The list of software functions in this appendix is based on responses to surveys performed as part of the preparation of this and previous handbooks. The first draft of this list was reviewed by representatives of this Air Force mission. However, it is possible that the list is not complete, or that another individual from the same organization would have described or categorized the software types differently. The number that follows each software function is the assigned software category. This software category is 1 of 18 standard categories defined in section 2.3.2 and it is used in path 1 to determine candidate methodology selection.

1. Threat Systems -- defensive and offensive radar, targeting/tracking systems, electronic countermeasures.

SOFTWARE FUNCTIONS	CATEGORY
controls and displays	14
message handling	8
radar range and position calculation	5
radar systems monitoring	10
servo positioning	10
target/threat data interpretation	16
tracking control	3

2. Guided Weapon Systems -- air-to-air, air-to-ground missiles, smart bombs.

SOFTWARE FUNCTIONS	CATEGORY
autopilot	6
built-in-test (BIT)	9
downlink telemetry	16

electronic counter-countermeasures	10
executive control	4
fusing	2
guidance	3
launcher sequencing	3
mission data preparation	12
navigation	5
tracking and stabilization	3

3. Scoring Systems -- proximity detectors for projectiles; used in aerial gunnery training.

SOFTWARE FUNCTIONS	CATEGORY
analog-to-digital conversion	13
front end formatting and filtering	13
graphics	14
performance statistics calculation	1
telemetry	13
trajectory calculation	5
trajectory calculation algorithms	5

APPENDIX B: AVIONICS

The software usage described in this appendix is based on a representative site devoted largely to this mission. Therefore, the procedures covered in this appendix may not include all aspects of the avionics mission.

B1. AERONAUTICAL SYSTEMS DIVISION (ASD)

ASD is a developing agency for weapons systems equipment, including avionics, automatic test equipment, crew training devices, and flight control and reconnaissance/C^3I systems. System and software development are typically contracted out; the development contractors tend to be medium to large size aerospace corporations, with substantial technical expertise in weapon systems development. The systems and embedded software are developed under well-defined contractual requirements and monitored by on-site representatives and frequent reviews of activity and documentation by ASD personnel. A wide diversity of software is developed by ASD, including numerous aircraft avionics and control systems, and communications systems software.

Development activities are controlled by Government standards and testing practices are fairly uniform, adhering to AF regulations and uniformly defined testing requirements, imposed by the SOW.

B2. ASD MISSION

The primary mission of Aeronautical System Division (ASD) is to acquire aeronautical systems that meet the needs of Air Force users such as Strategic Air Command, Tactical Air Command, Air Training Command, and Air Force Logistics Command to provide maintenance and support systems. Virtually all systems and equipments are developed by contractors, who are also responsible for development (and sometimes maintenance) of the computer programs and computer data.

Since there is very little weapon system software developed by ASD personnel, the main activities of ASD software engineers and software managers are to (1) assist in preparing the computer resource elements of specifications and requests for proposals, (2) participate in evaluating contractor proposals during the source-selection process, (3) monitor the progress and change activity during system development, and (4) assess the degree to which requirements are being satisfied. These software engineers and managers interact with other engineers and managers involved with the system and report their findings through appropriate ASD internal channels to program management for status, understanding, and decisions. In certain programs, Government personnel are assisted in their tasks by support contractors commonly called Systems Engineering and Technical Assistance and independent verification and validation (IV&V) contractors. IV&V

contractors are usually focused on software issues, while SETA contractors have a broader scope with software as one of the elements within that scope.

B3. SOFTWARE DEVELOPMENT ENVIRONMENT

This appendix covers five major types of weapon system software developed at ASD, including the development and mission support software associated with each type. These five types are categorized for convenience as avionics; automatic test equipment; air crew training device (ATD); flight control; reconnaissance; and command, control, communications, and intelligence (C^3I). The software developed by the ASD contractors is highly varied from category to category. Furthermore, the software within each category is far from homogeneous in its nature.

Avionics Software at ASD (Overview). Avionics software usually encompasses the software aboard aircraft, airborne strategic missiles, and some air-to ground missiles acquired by ASD. Aircraft avionics operational flight programs (OFP) are generally divided into two categories: (1) offensive avionics, including such functions as navigation; air data computation; weapons management; sensor data reduction and controls; stores management; target recognition and designation; cockpit controls and displays terrain avoidance; terrain following; computer executive functions; and communications; and (2) defensive avionics, including electronic threat detection; threat discrimination; threat avoidance; a variety of jamming techniques, controls, displays; and computer executive functions. The specific set of avionics functions depends on the nature of the aircraft (air-to-air fighter, air-to-ground fighter, multirole fighter, fighter bomber, strategic bomber, cargo, tanker, reconnaissance, or trainer) and the specific requirements for that aircraft.

On board automated built-in-test functions or central integrated test systems are used to determine hardware and system failures, to notify air crews for assessment of the effect on mission performance, and to notify ground crews for maintenance actions. These software functions may or may not be considered part of the avionics, depending on individual perspectives with ASD.

Detailed information regarding avionics software is contained in the following paragraphs.

a. Requirements. The software functional and performance requirements are generally derived by contractors from avionics system requirements of the same type. In addition, the Air Force may specify certain software design requirements that may change over the years, depending on the advancement of technologies.

   Generally, modern avionics software and firmware are distributed among special-purpose computers of the "minicomputer" class and among microprocessors.

Communication among processors is usually according to the protocol defined for serial multiplex buses in MIL-STD-1553B. The OFP is a real-time program usually operating under an executive concept of rigorously scheduled function execution in the foreground activities for each mode of the mission. The scheduling timeframe for foreground is a part of the design of the software, based on how frequently the required data are updated to achieve mission performance. Less critical functions operate in the background and are usually scheduled on a time-available basis, with higher priority activities interrupting those of lower priority. This rigorous scheduling is imposed to simplify the design concept and to ensure repeatability during software and system test so that transient anomalies are minimized.

b. Language. Until the B-1 program and the F-16 program, avionics software was exclusively programmed in assembly language. With the successful use of the JOVIAL language on the above two programs, the Air Force standardized on the JOVIAL J73 language (according to MIL-STD-1589B) for all avionics applications (unless an approved waiver based on technical or cost issues is granted). Offensive avionics software using JOVIAL usually has about 80% of the object code generated from JOVIAL source code with the remander in assembly code. Input/output functions (not supported by JOVIAL) are relegated to assembly code. With this 80/20 mix, the JOVIAL expansion of code and timing is estimated at 15% over that of well-done assembly code.

c. Size. The size of avionics OFP's varies considerably with application. For simple fighter aircraft, the OFP may be less than 3,000 instructions; whereas for a strategic bomber the total OFP size may be 100 times greater. As the functional requirements increase, so do the sizes of the OFP and its data tables. The Air Force usually requires a margin on sizing, timing, and communications throughput to provide for future modification and growth of the avionics suite. These margins may be initially specified in the range of 15% to 50% of total capacity, but the tendency has been for software growth to use a significant amount of this margin before development is complete.

d. Development facilities. Typically, avionics software is developed on a general-purpose host (IBM 370, VAX 11/780, etc.) in JOVIAL and initially targeted for the host. After error-free compilation of units, some minor checkout takes place on the host. Units are than recompiled on the host and targeted for the flight computer. Some unit and module testing is done through an instruction-level simulation or interpretive computer simulation on the host, but this is usually minimal because of long running time and slow turnaround. A software development laboratory (SDL) is used for module and computer program component checkout and integration. This laboratory simulates (usually on Harris type or VAX computers) other system hardware elements and drives the software in real time on breadboard processors. Each processor in the distributed system is at first driven "standalone" to test the software in that one computer.

A more extensive facility, the System Integration Laboratory and Test Facility (SILTF) is used by the software development team to integrate the various computer elements so that the distributed system may be exercised with as much real equipment (rather than simulated equipment) as is economical. After this phase, the software is handed over to a separate test team (within the same company), which conducts tests on the SILTF according to rigorous test plans and procedures developed by the contractor and reviewed by the Air Force. Digital and graphic data are recorded to verify correct functional performance against the verification cross-reference matrix contained in section 4.0 of the CPCI development specifications. Software problem reports or discrepancy reports are written by the software development team, and corrections prepared and periodically incorporated through contractor configuration control procedures. Retest is accomplished after corrections are implemented.

Test Equipment Software (Overview). There are three generic types of automatic test equipment that are procured by ASD for aircraft: flightline test equipment, intermediate shop test equipment, and depot test equipment. The purpose of the flightline ATE is to heck out aircraft systems to isolate faultly black boxes, or line replaceable units (LRUs). The intermediate shop test equipment designed for combat bases uses ATE to isolate faults in LRUs to specific electronic cards, or shop replaceable units (SRUs), which are then replaced. SRUs are either discarded or sent to one of five U.S. depots where the third type of ATE isolates the faulty components on the cards, which are then repaired and returned to inventory.

Detailed information follows regarding test equipment software.

a. Software categories. There are usually four categories of software that are part of the ATE software: (1) the operating system, which is usually provided by the vendor of the computer (often a commercially available machine); (2) the support software, which includes compilers or interpreters, assemblers, linkers, and loader (often commercially available); (3) control software, which controls the various electronic devices (signal generators and the like) that are part of the tester; and (4) unit-under-test (UUT) software, which activates in sequence the control software and measures the appropriate response of the UUT for that test condition. The UUT software also identifies the faulty elements of the UUT. The two most modern ATE developments are those for the F-16 aircraft and the Modular Automatic Test Equipment (MATE) program.

b. Language. The ATE software (for the first three categories above) is usually written in assembly language and FORTRAN, with the UUT software usually written in some version of ATLAS. The current Air Force standards are IEEE Standard C/ATLAS 716-1982 and 717-1982. The MATE program is an effort to standardize on ATE interfaces for future equipments. MATE is examining the use of JOVIAL J73 rather than FORTRAN for future ATE software and is advocating the use of

the above IEEE standards for ATLAS.

c. Development process. The requirements process usually starts with a Test Requirements Document (TRD) for the equipments to be tested by ATE. The TRDs are usually prepared by the designers or manufacturers of the units. From the TRDs and previous experience, a weapon system contractor or an ATE contractor derives or selects the test station requirements, which include the operating system and control software requirements. From the TRDs and detailed documentation on the UUTs and test stations, the specifications for UUT software are derived. In more recent systems, these documents have been reviewed by IV&V contractors, which the SPOs have indicated to be beneficial and cost-effective.

d. Testing. For economic reasons, the tests to be implemented in ATE software can never be totally exhaustive. There are many failure modes possible to a UUT, either singly or in combination; consequently, those that constitute a high percentage (90% to 95%) of all failures are implemented in the UUT software.

It is not economical to test every fault isolation option in UUT software on actual test hardware. Since the fault itself must be inserted into the UUT to conduct the test (this may be difficult to do without inserting multiple faults) and since the number of test units available may be limited, UUT software testing usually is slow and expensive. Usually for tests witnessed by the Air Force, 50 to 100 different tests are run on the software against a single, actual UUT. Remaining errors, problems, or needed test programs are resolved during the software maintenance activity, with some economic justification.

e. Special requirements. Fault tolerance is seldom a requirement for ATE software. If there is a software fault in the tester or the UUT, the philosophy has been that the fault should be repaired rather than provide software to work around that fault. Self-test is usually provided in the test station so that test station hardware failures may be isolated and repaired quickly to bring the test station back on line.

f. Development environment. Modern systems normally compile, interpret, and assemble on the test computer. An exception to this is the F-15 intermediate shop test equipment system, which compiles the UUT software on an IBM 360. This approach is now deemed to be less efficient. Much of the debugging is done on the tester itself with a real hardware UUT in place, turn-around through a separate host takes too long, and the minicomputer in the test set has the capacity to do the hosting job.

Neither environmental simulators nor simulations of the UUT are used. The first is not needed and the second is not considered effective. Writing and debugging the simulation of the UUT is considered more expensive than the present methods that use the actual UUT hardware.

Simulator Software (Overview). ASD acquires a variety of automated crew training devices, ranging from simple part-task trainers, which provide a training element that may be as short as 8 to 10 minutes duration, through full weapon system trainers, which may simulate an 8 to 10 hour mission for an entire strategic bomber crew, including pilot, copilot, navigator, flight engineer, and electronics warfare officer. For pilot training, these air crew training devices present all of the cockpit instruments and displays, pilot controls, window and heads-up displays, motion effects, aural cues and effects, realistic aerodynamic response for the simulated aircraft, engine responses, vibration, and avionics equipment behavior, including sensor behavior and weapon release.

a. Virtually all of the functions are simulated in software on one or multiple commercially available 32-bit minicomputers. In a recent system (F-16), a copy of the aircraft avionics computer with its flight software has been included in the simulator itself, rather than simulating the flight programs on a general-purpose computer. Commercially available operating system software, peripheral devices (tapes, disks, printers, CRTs, etc.) and their control software are generally used. Applications programs that simulate the aircraft function and perform much of the instructor station operation are specified to be written in FORTRAN.

b. Software characteristics. The fundamental control philosophy for simulator software design is a prescheduled, synchronous timeframe for those highly cyclic aircraft activities with other lower priority activities running in the background on an interruptible basis. Modularity (one function per module), top-down design, and separation of data from program instructions are usually requirements on the software effort. The software generally checks the ranges and validity of instructor-supplied parameters and provides fault data for maintenance purposes.

c. Specifications. Usually the entire ATD is a single configuration item of which the software is an element. The system specification for the ATD indicates the functions to be represented in the system, the general level of fidelity, the required margins for computational speed, bus throughput, directly addressable memory, and bulk memory (usually disk).

d. Testing. The fundamental adequacy of the software and simulator performance is judged through formal tests in which experienced pilots aircraft. Other elements of the system, such as the instructor station capabilities, are verified by objective tests.

Flight controls (Overview). Digital flight controls and digital engine controls represent relatively new areas of computer application at ASD. Both involve the primary issues of high performance and flight safety. Whereas the computer resource activity for the three applications previously discussed have had at least 10 to 15 years of history and evolution at ASD, these control applications are relatively new and are not yet implemented in an aircraft scheduled for production.

a. Language. Development of flight control software has to date been in assembly language but will no doubt be done in HOL in the future as mature compilers become available. The general development and test procedures are similar to those for avionics software.

b. Software characteristics. The control law implementation for multiaxis stability and aircraft control are highly algorithmic in nature with different algorithms and different control gains for different flight modes or regimes. The software is written against prescheduled time increments so that periodic data updates and control computations are completed at a cyclic frequency to maintain an adequate margin for stability and control. Less important functions are scheduled in the background, some of which may be triggered on an event rather than on a cyclic basis.

   The preparation of flight control software requires a thorough understanding of the hardware implementation, both in its failed states and its unfailed states, as well as an understanding of control theory. The testing of this software is complicated by the requirement to test the system both in its nominal state and in its large number of failure combinations.

c. Testing. ASD/ENF is currently determining what will be necessary in the software requirements and test area for safety qualification. The character of flight control software can be generally ascertained by a review of the advanced fighter technology integration (AFTI) programs being pursued by the Flight Dynamics Laboratory in the Air Force Wright Aeronautical Laboratories.

d. Special requirements. In general, the sensor and computational hardware will be triple or quadruple redundant so that failure of one or two processors would not jeopardize flight safety. Sensor data will be cross-strapped among the processors so that each can operate on the full set of sensor data with identical software. Comparison of input data from similar sensors will be used to isolate failed sensor strings. Comparison of output data will be used to identify failed processor elements. The fault-tolerance requirement (hardware fault detection and isolation) is a key requirement and adds considerable complexity, particularly if battle damage causes aerodynamic and control surface changes.

Overview of Reconnaissance and C^{3}I. ASD is responsible for the acquisition of ground systems that receive (from aircraft sensors) data regarding the ground threat environment. These data, such as radar digital maps or ground electronic emissions information, are processed to determine the nature and location of various threats. Upon threat identification and location (during a real battle), the ground computational system, in conjunction with human controllers, may (1) plan an action against some of the threats, (2) allocate weapon and aircraft resources, and (3) control the flightpath of the aircraft and/or its weapons to the vicinity of the selected targets.

a. Hardware. These systems may be implemented in commercially available or militarized versions of commercial computers. These are usually of the mini or super minicomputer class with special-purpose, high throughput processors for signal processing and distributed, tightly coupled parallel processors for the remainder of the processing.

b. Software. FORTRAN and assembly are usually the languages used for the application software that is structured and modular. The development is usually on the minicomputers used for the project, and the testing procedures parallel those used in the SILTF.

Future flight control software will interact with the avionics software (1) to achieve automated delivery of weapons and (2) to use avionics sensors. A key requirement on this type of software will be that errors in the avionics system shall not propagate into the flight control software. This may be a difficult requirement to validate.

B4. CATEGORIZATION OF SOFTWARE FUNCTIONS

The list of software functions in this appendix is based on responses to surveys performed as part of the preparation of this handbook. The first draft of this list was reviewed by representatives of this Air Force mission. However, it is possible that the list is not complete, or that another individual from the same organization would have described or categorized the software types differently.

The number that follows each software function is the assigned software category. This software category is 1 of 18 standard categories defined in section 2.3.2 and is used in path 1 to determine candidate methodology selection.

A. Airborne Systems

1. Avionics Systems

a. Mission Avionics

SOFTWARE FUNCTIONS	CATEGORY
aerial delivery	3
automatic approach/landing	4
communication	10
control/display processing	14
data bus control	2

navigation/guidance	5
real time executive	2
self-test	9
sensor control	3
sensor data reduction	10
sensor test/credibility	10
terrain following/avoidance	2,16

b. Offensive Avionics

SOFTWARE FUNCTIONS	CATEGORY
stores management	1
target recognition/acquisition	16
weapons control	3

c. Defensive Avionics

SOFTWARE FUNCTIONS	CATEGORY
jamming	10
threat avoidance	2
threat detection	10
threat discrimination	16

2. Flight Critical Systems

a. Flight Control

SOFTWARE FUNCTIONS	CATEGORY
digital flight control	3
sensor data processing	10

b. Fuel Management

SOFTWARE FUNCTIONS	CATEGORY
fuel management	2

c. Engine Control

SOFTWARE FUNCTIONS	CATEGORY
digital engine control	3
engine cycle data acquisition	13
fault detection and accommodation	2,9

B. Ground Systems

1. Air Crew Training Devices

SOFTWARE FUNCTIONS	CATEGORY
aural system	3
computation system (executive, support, maintenance software)	1,4
digital radar land mass system	3
electronic warfare system	3
electro-optical viewing system	3
gravity seat systems	3
instructor/operator station	14
motion systems	3
student station	2
visual system	14

2. Automatic Test Equipment

SOFTWARE FUNCTIONS	CATEGORY
control software	2
system software (compilers, support)	17
unit under test (UUT) software	9

APPENDIX C: COMMAND, CONTROL, COMMUNICATIONS, AND INTELLIGENCE

The software usage described in this appendix is based on a representative site devoted largely to this mission. Therefore, the procedures covered in this appendix may not include all aspects of the command and control mission.

C1. STRATEGIC AIR COMMAND

The Strategic Air Command (SAC) has a diversity of missions to support, such as command and control, war planning, intelligence support, and strategic weapons support. It also develops a wide diversity of unrelated systems for these missions. For strategic weaponry, SAC is a user agency, while for other areas it is both a developer and user. War planning and intelligence systems are developed and maintained almost exclusively by Air Force personnel, while often the development of information and mangement systems are primarily conducted by contractors with the maintenance shared by Air Force and contractor personnel. The software developed for the warning functions ranges from highly critical to noncritical. Software development practices for contractors are controlled by the SOW; internal maintenance is conducted in accordance with SAC regulations. SAC computation systems tend to be data base and data processing intensive, such as in the intelligence and war planning areas. The warning area includes real-time control functions, and the command centers use C^3I technology software. SAC-conducted software testing practices and methods are standardized by SAC regulations; however, there exists variability in their application, corresponding to the differences in the software categories, criticality, and functional organizational practices.

C2. SAC MISSIONS

The missions performed by SAC include command and control, war planning, warning and intelligence support. The general functions performed within these missions are as follows.

Automated command control:

a. Collection of status-of-forces information on a near-real-time basis, using generalized information on a near-real-time basis and a generalized software system called the Force Management Information System.

b. All geographically dispersed SAC subordinate units are linked to headquarters computers via a Data Transmission Subsystem.

c. Command Post wall screen and printer displays provide data to the Commander-in-Chief Strategic Air Command (CINCSAC) and the Battle Staff concerning availability of resources for Single Integrated Operational Plan (SIOP) execution.

Progress of force activity can be monitored as events materialize.

d. Support is provided to the Single Tanker Missions for worldwide Tactical Air Command (TAC) aircraft deployments.

e. Support is provided to the SAC aircraft contingency planning staff.

f. Software development support is provided for Numbered Air Force control Systems.

g. Software development support is provided for Airborne Command Post Force control Systems.

War planning:

a. Planning of intercontinental ballistic missiles (ICBM), aircraft, and cruise missile sorties against specified enemy targets is accomplished using intelligence estimates, weapons capabilities, and geological factors.

b. Computer simulations permit "flying" sorties to determine success probability.

c. Extensive use of interactive graphics permit SAC and JSTPS planners to visualize SIOP development.

d. Production of flight plan cassettes for unmanned cruise missiles.

e. Gaming techniques provide information on methods to improve the plan by pitting the SIOP against the probable enemy plan.

Warning:

a. Computers embedded in various missile warning field sensors enable the detection and/or tracking of hostile missile launches.

b. Near-real-time displays on several display devices notify Command Post personnel of endangered SAC resources and provide information needed for decisions of force posturing, including launch for survival of aircraft.

c. An automated countdown to impact and checklists of required actions greatly assist the decisionmaking process.

Intelligence support:

a. Online interactive analyst support is provided for collection management, photographic and electronic intelligence analysis and correlation, target development for the National Target Base and maintenance of offensive and defensive orders-of-battle on the SAC On-Line Analysis and Retrieval System (SOLARS).

b. Automated processing of electronic intelligence (ELINT) is accomplished to support the airborne reconnaissance program.

c. Development of processing systems provide for SC evaluation of airborne reconnaissance collectors.

d. Use of graphic displays supports processing of scientific and technical data describing electronic emitter characteristics.

e. Map overlay plotting is used to support SIOP and ELINT production.

f. Automated support to reprogrammable airborne electronic warfare systems is provided.

g. Communications support and online analytical support for operational intelligence analysts are provided.

Management support:

a. The management information requirements of the HQ SAC staff are supported with 40 Air Force standard and 39 command-unique Management Information Data Systems.

b. Remote terminals in te Headquarters building permit online support.

c. Computer output microfiche (COM) capability is available, as well as the Honeywell Page Printing System.

d. Liaison is maintained with the Air Force Data Systems Design Center, manpower and Personnel Center, Accounting and Finance Center, and other MAJCOMs.

C3. CATEGORIZATION OF SOFTWARE FUNCTIONS

The list of software function in this appendix is based on responses to surveys performed as part of software test handbook preparation, updated by a later survey during preparation of these guidelines. This list has been reviewed by representatives of this Air Force mission. It is possible, however, that the list is not complete, or that another individual from the same organization would have described or categorized the software types differently.

The number that follows each software function is the assigned software category. This software category is 1 of 18 standard categories defined in section 2.3.2 and is used in path 1 to determine candidate methodology selection.

SOFTWARE FUNCTIONS	CATEGORY
controls and displays	14
data base management	12
interactive interface	14
mapping/plotting (graphics)	14
mission data preparation	12,1
sensor data processing	10
simulation (non-real-time)	11
simulation (real-time)	11
tracking	6

APPENDIX D: SPACE

The software usage described in this appendix is based on a representative site devoted largely to this mission. Therefore, the procedures covered in this appendix may not include all aspects of the command and control mission.

D1. SPACE DIVISION

The Space Division (SD) is a development agency for space-related systems, including satellites, launch vehicles, and ground control and communications systems. SD relies extensively on contractors to develop its systems and embedded software, which also performs maintenance under follow-on contracts. Software development practices for contractors are controlled by the SOW; and SD personnel, often coupled with technical consultant contractors, monitor all development activities at all levels intensively. Frequent reviews and technical direction are provided by this agency. A wide diversity of software categories is developed by SD, including software for communications, satellite control systems, prelaunch checkout and ground test systems, space vehicle avionics and control, and system simulations. This site employs IV&V contractors to a greater extent than any of the other sites surveyed. Software development practices are established by Air Force regulation, defined by SOW, and, as a result, tend to be relatively uniform among the development contractors. SD places great emphasis on the thoroughness, sufficiency, and formality of contractor development practices.

D2. SD MISSION

The primary mission of SD is to acquire space-related systems, which include satellites, launch vehicles, and ground control and communications systems. Also, SD is responsible for managing and operating some elements of these acquired systems, such as the Satellite Control Facility and the Vandenberg Launch Facility. The new Space Command will impact these operating activities in a way that is yet to be determined.

SD relies on contractors to develop its systems, including the software within its systems. Development contractors for SD usually continue to maintain the software (if maintenance is required).

Because systems and system software are not developed by SD, the main activity of its personnel is to prepare RFPs, evaluate proposals, and conduct software management surveillance during the contract.

Technical assistance in the software area is provided by Aerospace Corporation, a nonprofit systems engineering and technical direction contractor, providing technical consultation to the Air Force. In addition, particular major programs are usually technically assisted by IV&V contractors, who are selected competitively on a program-by-program basis.

D3. SOFTWARE DEVELOPMENT ENVIRONMENT

This section of the report will present an overview of four major types of SD software; ground control and communications systems, prelaunch checkout and launch systems, launch vehicle systems, and space vehicle systems. It will be evident that the software developed by SD contractors is highly varied in character from category to category. There is no technical detail concerning SD software that is true for all applications at this site. Following these overviews, the report will focus primarily on those specific applications encompassed in the survey.

SD relies on contractors using their own tools to develop and test software. Unlike the aeronautical systems, space systems rely on development contractors to continue to maintain the software through its life cycle; furthermore, for certain systems a high degree of contractor IV&V is provided. The software characteristics within an entire system and across systems vary substantially.

Ground Control and Communications (Overview). SD has acquired and is acquiring systems and software that (1) control the attitudes and functioning of unmanned satellites; (2) gather, reduce, record, and display data transmitted by satellites; and (3) communicate and control manned space activities. Some examples of these systems include the Satellite control Facility (SCF) at Sunnyvale, the Global Positioning Satellites ground component, and the consolidated Space Operations Center (CSOC) at Colorado Springs.

These systems have extensive amounts of software with major functions such as satellite detection (both enemy and friendly), orbit determination, displays of status to controllers, and satellite attitude correction and communication with one another and remote sites. In fact, if either the CSOC or the SCF become disabled, they can perform each other's functions (as can the Johnson Space Flight Center).

Most of the software is written in higher order language (JOVIAL and HAL-S with some FORTRAN for CSOC and JOVIAL for SCF) and operates in near-real time. CSOC software is maintained by Air Force personnel with substantive contractor support, while SCF primarily uses contractor maintenance. These systems are generically similar to SD's reconnaissance and C^3I systems but are larger in scope and more multipurpose.

Prelaunch Checkout and Launch System (Overview). A considerable investment in software resides in prelaunch checkout and ground launch systems that perform the booster and satellite ground checks before and during countdown and that perform the range safety function during launch. For new vehicles, the existing facilities are adapted, new software written, and additional factory checkout equipment moved to the launch site.

The software requires a detailed understanding of the hardware being checked and the system's function. These activities bear a similarity to SD's automatic test equipment

software, but on a more focused scale since the checkout activity usually is confined to the contractor's facility and the launch site.

Launch Vehicle Systems (Overview). Software in launch vehicle systems maintains the stability of the vehicle during its flyout and takes inertial and other sensor information in order to follow a preplanned launch trajectory. The software design is based on a rigidly scheduled, cyclic sequence of events much like aircraft avionics software. This software is usually small in program size, often fitting into a 16K-word memory and is usually written in assembly language.

The development and testing of this software parallels that described for SD's avionics software, except that it is simpler in function for the vehicles that launch unmanned payloads.

Space Vehicle Systems (Overview). Space vehicle systems may be classified as satellites that are manned, such as the Space Shuttle, and unmanned vehicles that are used for exo-atmospheric transport, such as the inertial upper stage (IUS), and the Mini Vehicle used in the Antisatellite System.

For the most part, manned satellites do not use digital computers, aside from some recent systems that have small processors for attitude sensing and pointing and other station-keeping responsibilities. More use of digital computers in future satellites is expected, with emphasis for these computers on low power and fault-tolerant design.

The Space Shuttle has substantial onboard software, but this effort was primarily a NASA effort and beyond the scope of this study.

Exo-atmospheric transport vehicles again are very similar to launch vehicles in their software natures, with the exception of the additional feature of engine control, more extensive maneuvering, and payload dispensing. Again, like aircraft avionics, simulation using a hot bench (SILTF-like facility) is performed during the design and testing to establish the real-time performance.

Usually, more extensive IV&V is performed on these systems than on SD system. This IV&V usually includes independent testing on separate facilities, using separate tools by IV&V contractors.

D4. CATEGORIZATION OF SOFTWARE FUNCTIONS

The list of software functions in this appendix is based on responses to surveys performed as part of software test handbook preparation, updated by a later survey during preparation of these guidelines. This list has been reviewed by representatives of this Air Force mission. It is possible, however, that the list is not complete, or that another individual from the same organization would have described or categorized the software types differently.

The number that follows each software function is the assigned software category. This software category is 1 of 18 standard categories defined in section 2.3.2 and is used in path 1 to determine candidate methodology selection.

A. Equipment Checkout -- pre-launch checkout, equipment self-test

SOFTWARE FUNCTIONS	CATEGORY
automatic test equipment (ATE)	9
built-in-test (BIT)	9
central integrated test systems	9

B. Aerospace Defense -- threat detection and warning, threat evaluation

SOFTWARE FUNCTIONS	CATEGORY
automatic processing	13
data base management	12
filtering and smoothing	9
guidance and control	3
message processing	8
mission data preparation	12
mission planning	15
real-time control	2
real-time executive	2
satellite impact prediction	5,7
satellite tracking	5
sensor processing	10
(sensor) tracking	5
simulation	11
situation notification	14
space information correlation	1
space situation analysis	15
task selection and displays	14

APPENDIX E: MISSION/FORCE MANAGEMENT

The software usage described in this appendix is based on a representative site devoted largely to this mission. Therefore, the procedures covered in this appendix may not include all aspects of the command and control mission.

E1. TACTICAL AIR COMMAND

The Tactical Air Command (TAC) is the development and user agency for the major Air Force tactical planning system, Computer Assisted Air Force Management System (CAFMS). The CAFMS is a single-function, highly interrelated automated processing system. The major output product of CAFMS is the air tasking order report. CAFMS was developed by TAC personnel with some contractor assistance during the early requirements and design phases. Management, development, and maintenance of this system are well defined and uniquely adapted for its ongoing support. The system is currently operational, but undergoes continual enhancements and incorporation of new capabilities. The overall function of the CAFMS is quite critical, but few of its software components are considered to be more than moderately critical. The system does incorporate some automated fallback provisions in case of failure, but redundancy of systems function is not provided and reversion to manual operation is the ultimate fallback provision. Testing practices are well defined and are incorporated as an integral part of a version release management system developed by TAC specifically for CAFMS. Testing is applied uniformly to all software components undergoing development. to the differences in the software categories, criticality, and functional organizational practices.

E2. TAC MISSIONS

The Tactical Air Command operates the Tactical Air Control Centers (TACC). The mission of TACC is to prepare, issue, and monitor the execution of coordinated orders for the employment of all forces available to the Air Force Component Commander. The TACC is the operation center of the Tactical Air Control System (TACS). The CAFMS was developed to augment the TACS with automated information processing, storage, and display capabilities, and secure digital communications capabilities.

The primary mission area applicable to the TAC is Mission/Force Management. An applicable secondary mission area is C^3I, because the Mission/Force Management functions are integrated into and communicates through a communications network.

E3. SOFTWARE DEVELOPMENT ENVIRONMENT

This section provides a summary of CAFMS and its software development environment. TAC has no other tactical systems involving significant software development or maintenance that were applicable to the survey. All elements of CAFMS are developed, programmed, and controlled in the same manner and within the same organizational

structure. The CAFMS is essentially a heterogeneous system in this respect. All code is implemented and tested according to the same standards and procedures. Therefore, the CAFMS was the only system surveyed for TAC. It is discussed as a single entity in this report, although in actuality it comprises a number of individual but interrelated computer programs. Each of these programs is developed by a uniform and disciplined management process.

Virtually all software effort on CAFMS is considered to be new development, as opposed to maintenance of existing program elements. This development effort involves augmenting the existing system with additional functions and integrating them into the overall design. It also involves major revisions to the system performance parameters, such as the data base contents, to enhance the system or to accommodate new functions. In this manner, the CAFMS is undergoing an evolutionary development process to meet current tactical planning demands and also to adapt it to changes in its operational environment. All changes are accomplished in a phased approach.

Development of CAFMS requirements was shared about equally between Air Force personnel and a supporting contractor. Design and development through initial installation were accomplished mainly by the Air Force, with only about 10% done under contract. Subsequent development and maintenance are entirely the responsibility of TAC. The system is currently undergoing initial operational test and evaluation.

Criticality factors for CAFMS include major mission impact, which probably is representative of Air Force mission/force management systems. The confidence level (see table B-1 in appendix B, for explanation) that applies to software development and testing is level 2. Therefore, the development disciplines and level of software error detection are comparable to many of the other major Air Force weapon systems, such as command and control and avionics systems.

CAFMS Overview. CAFMS is designed primarily to build, disseminate, and monitor the execution of the Air Tasking Order (ATO). There is also a requirement to build and generate a variety of status reports and periodic and end-of-day summaries. CAFMS reduces ATO preparation time. Since TACC is mobile, CAFMS must be capable of limited deployment. Therefore, there must be some ability to identify and change the names, locations, etc., of subelements in the data base. Also, CAFMS must be capable of processing classified information up to and including SECRET. CAFMS provides an automated assist to the manual system for some of its key functions. The main operating centers are the 9th Air Force, and the USAF Tactical Air Warfare Center. CAFMS is intended to fulfill the following requirements.

a. Increase capacity and accuracy in the display of air situation and mission progress data.

b. Maintain status of bases and forces.

c. Significantly decrease the time required for preparation and dissemination of the ATO.

d. Significantly decrease the time used in routine and clerical tasks associated with mission planning.

e. Automatically generate and disseminate status and summary reports.

f. Provide terminals at the Control and Operations Centers, Air Support Operations Centers, Wing Operations Centers, and TACC.

g. Maintain status of communications, weather, munitions, etc.

h. Provide an offline AUTODIN interface from the TACC to any AUTODIN user, through the 470L System, TACS Communication System.

System Description. CAFMS has the following six major system functions:

a. Startup. The startup function initializes all other system functions during initial startup or during recovery. This initialization includes establishment of the system environment; for example, communications assignments for participating units, message alert routing, display access authorization, and system access authorization. The data base is initialized either to start clean or, if after a recovery, to start at the last saved position. Communications initialization facilitates hookup of all remote terminals and other communications links.

b. Console. At TACC, the console functions include the ability to build, update, and disseminate the ATO. It also includes the automatic building of mission schedule files to be used by current operations and report generation for each day's activities. Console functions common to both the remote terminals and the TACC include log-in to gain access to the system, display printing capability, review of the ATO, update and delete capabilities for mission schedule and other files, input validation, and the display function itself.

c. Communications. CAFMS communications function provides the interface between the TACC and external elements not equipped with a remote terminal. This offline capability allows dissemination of messages (primarily ATO) through AUTODIN or the TACS internal teletypewriter (TTY) network.

d. System environment definition. This function provides the capability to maintain and change or update the system environment as necessary. This includes a capability to receive a printed listing of any specified system environmental data (e.g.,

message routing table).

e. Message processing. The message processing function provides the capability to prepare the JINTACCS ATO display formats for transmission to addressees not possessing a remote terminal. This conversion process or reformatting includes the insertion of header and trailer information. When the message has been formatted, it is stored in a message file and later output to the offline paper tape punch.

f. Shutdown. The shutdown function provides the capability for either an orderly termination of all computer system functions or, if necessary, an emergency termination. An orderly shutdown includes notification to all consoles and remote terminals that shutdown has started. All messages queued to the paper tape punch are completed. The system environment and necessary data base information are saved, as well as any recording information being generated. In accordance with appropriate security directives, memory and disk are overwritten. In the case of an emergency shutdown, only the memory and disk overwrite function are accomplished.

System Data Characteristics. For in-garrison operations, external data inputs are received by voice communications to the TACC. These data are manually entered into the system through local consoles. In deployed operations, inputs are provided through the remote terminals and/or voice communications. Functional user data inputs are as follows:

- Aircraft/Aircrew Status
- Munitions Status
- Weather Status
- Unit/Base status
- Air/Ground Situations
- Communications Link Status

The data outputs provided by CAFMS are the ATO message, Mission Schedule displays, and Status/Report displays. The following displays are available in CAFMS:

- Air Tasking Order

- Mission schedule Displays
    - Fighter/FAC/Support/Other
    - Reconnaissance
- Status/Report Displays
    - Unit/Base Status
    - Aircraft/Aircrew Status
    - Munitions Status
    - Weather Status
    - Aircraft Losses
    - Unit Air Sortie Recap
    - Mission Air Sortie Recap
    - Communication Circuits Status
- Strike packages

Standards and Documentation. The major regulations applicable to CAFMS software development are the AFR 300 series and AFR 800-14, and DoD 7935.1-S, Automated Data Systems Documentation Standard. Applicable computer program documentation includes the following items:

- System specification
- Computer program design specification
- Configuration management plan
- Data base specification
- Operator's manual
- User's manual

- Functional description
- Development test plan (one per module)

Programming standards and conventions identified for CAFMS provide coverage for top-down structured development (analysis, design, and process), coding standards and testing requirements (module, subsystem, and system testing).

E4. CATEGORIZATION OF SOFTWARE FUNCTIONS

The list of software functions in this appendix is based on responses to surveys performed as part of software test handbook preparation, updated by a later survey during preparation of these guidelines. This list has been reviewed by representatives of this Air Force mission. It is possible, however, that the list is not complete, or that another individual from the same organization would have described or categorized the software types differently.

The number that follows each software function is the assigned software category. This software category is 1 of 18 standard categories defined in section 2.3.2 and is used in path 1 to determine candidate methodology selection.

SOFTWARE FUNCTIONS	CATEGORY
communication	10
controls and displays	14
data base management	12
mapping/plotting (graphics)	14
message processing	8
secure data processing	12,8,4
war planning	15

APPENDIX F: MISSILES

The software usage described in this appendix is based on a representative site devoted largely to this mission. Therefore, the procedures covered in this appendix may not include all aspects of the command and control mission.

F1. BALLISTIC MISSILE OFFICE

The Ballistic Missile Office (BMO) is the responsible agency for ballistic missile systems, including launch vehicles, and ground control and communications systems. BMO relies extensively on contractors to develop its systems and embedded software, which also performs maintenance under follow-on contracts. Software development practices for contractors are controlled by the SOW; and BMO personnel, often coupled with technical consultant contractors, monitor all development activities at all levels intensively. Frequent reviews and technical direction are provided by this agency. A wide diversity of software categories is developed by BMO, including software for communications, missile control systems, prelaunch checkout and ground test systems, missile vehicle avionics and control, and system simulations. This site employs IV&V contractors to a great extent. Software development practices are established by Air Force regulation, defined by SOW, and, as a result, tend to be relatively uniform among the development contractors. BMO places great emphasis on the thoroughness, sufficiency, and formality of contractor development practices.

F2. BMO MISSION

The primary mission of BMO is to acquire ballistic missile-related systems, which include missiles, missile launch vehicles, and ground control and communications systems. Also, BMO is responsible for managing and operating some elements of these acquired systems, such as the Satellite Control Facility and the Vandenberg Launch Facility. The new BMO will impact these operating activities in a way that is yet to be determined.

BMO relies on contractors to develop its systems, including the software within its systems. Development contractors for BMO usually continue to maintain the software (if maintenance is required).

Because systems and system software are not developed by BMO, the main activity of its personnel is to prepare RFPs, evaluate proposals, and conduct software management surveillance during the contract.

Technical assistance in the software area is provided by non-profit systems engineering and technical direction contractors, providing technical consultation to the Air Force. In addition, particular major programs are usually technically assisted by IV&V contractors, who are selected competitively on a program-by-program basis.

F3. SOFTWARE DEVELOPMENT ENVIRONMENT

This section of the report will present an overview of four major types of BMO software; ground control systems, prelaunch checkout systems, launch vehicle systems, and missile systems. It will be evident that the software developed by BMO contractors is highly varied in character from category to category. There is no technical detail concerning BMO software that is true for all applications at this site. Following these overviews, the report will focus primarily on those specific applications encompassed in the survey.

BMO relies on contractors using their own tools to develop and test software. Unlike the aeronautical systems, missile systems rely on development contractors to continue to maintain the software through its life cycle; furthermore, for certain systems a high degree of contractor IV&V is provided. The software characteristics within an entire system and across systems vary substantially.

Ground Control Systems (Overview). BMO has acquired and is acquiring systems and software that control manned missile activities.

These systems have extensive amounts of software with major functions such as threat detection, navigation and guidance, displays of status to controllers, and course correction and abort from remote sites.

Most of the software is written in higher order language (JOVIAL and HAL-S with some FORTRAN and JOVIAL) and operates in near-real time. Some software is maintained by Air Force personnel with substantive contractor support, while the remainder uses contractor maintenance.

Prelaunch Checkout Systems (Overview). A considerable investment in software resides in prelaunch checkout and ground launch systems that perform the missile ground checks before and during countdown and that perform the range safety function during launch. For new vehicles, the existing facilities are adapted, new software written, and additional factory checkout equipment moved to the launch site.

The software requires a detailed understanding of the hardware being checked and the system's function. These activities bear a similarity to BMO's automatic test equipment software, but on a more focused scale since the checkout activity usually is confined to the contractor's facility and the launch site.

Launch Systems (Overview). Software in launch vehicle systems maintains the stability of the vehicle during its flyout and takes inertial and other sensor information in order to follow a preplanned launch trajectory. The software design is based on a rigidly scheduled, cyclic sequence of events much like aircraft avionics software. This software is usually small in program size, often fitting into a 16K-word memory and is usually written in assembly language.

Missile Systems The development and testing of this software parallels that described for avionics software, except that it is simpler in function for the ballistic missiles.

Again, like aircraft avionics, simulation using a hot bench (SILTF-like facility) is performed during the design and testing to establish the real-time performance.

Usually, extensive IV&V is performed on missile systems. This IV&V usually includes independent testing on separate facilities, using separate tools by IV&V contractors.

F4. CATEGORIZATION OF SOFTWARE FUNCTIONS

The list of software functions in this appendix is based on responses to surveys performed as part of software test handbook preparation, updated by a later survey during preparation of these guidelines. This list has been reviewed by representatives of this Air Force mission. It is possible, however, that the list is not complete, or that another individual from the same organization would have described or categorized the software types differently.

The number that follows each software function is the assigned software category. This software category is 1 of 18 standard categories defined in section 2.3.2 and is used in path 1 to determine candidate methodology selection.

A. Equipment Checkout -- pre-launch checkout, equipment self-test

SOFTWARE FUNCTIONS	CATEGORY
automatic test equipment (ATE)	9
built-in-test (BIT)	9
central integrated test systems	9

B. Missile Defense -- threat detection and warning, threat evaluation

SOFTWARE FUNCTIONS	CATEGORY
automatic processing	13
data base management	12
filtering and smoothing	9
guidance and control	3
message processing	8
mission data preparation	12
mission planning	15
real-time control	2
real-time executive	2
sensor processing	10
(sensor) tracking	5
simulation	11
situation notification	14
missile information correlation	1
missile situation analysis	15
task selection and displays	14

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.

END

FILMED

2-86

DTIC